WISE-Flow: Workflow-Induced Structured Experience for Self-Evolving Conversational Service Agents

Yuqing Zhou[†1] **Zhuoer Wang**[2] **Jie Yuan**[2] **Hong Wang**[2] **Samson Koelle**[2]
Ziwei Zhu[1] **Wei Niu**[2]
[1]George Mason University [2]Amazon
{yzhou31, zzhu20}@gmu.edu, {wze, niuwei}@amazon.com

Abstract

Large language model (LLM)-based agents are widely deployed in user-facing services but remain error-prone in new tasks, tend to repeat the same failure patterns, and show substantial run-to-run variability. Fixing failures via environment-specific training or manual patching is costly and hard to scale. To enable self-evolving agents in user-facing service environments, we propose WISE-Flow, a workflow-centric framework that converts historical service interactions into reusable procedural experience by inducing workflows with prerequisite-augmented action blocks. At deployment, WISE-Flow aligns the agent's execution trajectory to retrieved workflows and performs prerequisite-aware feasibility reasoning to achieve state-grounded next actions. Experiments on ToolSandbox and τ^2-bench show consistent improvement across base models.

1 Introduction

By augmenting large language models (LLMs) with external tools and an iterative action-observation loop, LLM-based agents can resolve multi-step tasks beyond text generation (Lewis et al., 2020; Schick et al., 2023; Yao et al., 2022). Such agents are increasingly deployed for task-oriented user-facing services. However, they can still make mistakes in practice, especially on new tasks or unfamiliar environments, repeatedly fail in similar ways, and exhibit high variance across runs. Fixing these failures via environment-specific training or manual patching is costly and hard to scale. This motivates self-evolving agents that improve over time by themselves (Gao et al., 2025).

A key aspect of self-evolution is the ability to learn from past experiences (Gao et al., 2025). Recent work has studied memory-augmented agents that store experience from prior interactions and retrieve it to guide future decisions. Abstract summary memories capture high-level summaries or reflections (Zhong et al., 2024; Shinn et al., 2023). While they provide general takeaways, they often remain **free-form and abstract**, providing limited support for deriving concrete, executable procedures, especially in the early stages of a task. Step-level trace memories store fine-grained traces, such as individual tool calls or events (Zhang et al., 2023; Xu et al., 2025). Although useful for recalling specific past steps, these traces are often **fragmented and lack procedural structure**, e.g., an explicit task-level workflow with prerequisites and recovery paths. Distillation methods further improve usability by converting offline trajectories into compact guidelines (Fu et al., 2024) or workflows (Wang et al., 2025b). However, distilled guidelines are often **too coarse to align with the agent's live progress** and yield the next feasible action under the current tool state. Even when workflow memories include step-level state descriptions, they typically **do not encode action prerequisites** as explicit, checkable constraints for execution-time validation. Collectively, these limitations leave a persistent retrieval-to-action gap. A complete discussion of related work is in Appendix A.1.

To address the aforementioned gaps, we propose Workflow-Induced Structured Experience (WISE-Flow), a workflow-centric framework for self-evolving agents, which turns past experiences into reusable procedural experience rather than isolated steps or free-form reference. WISE-Flow provides execution-time guidance for conversational agents in user-facing services. This guidance takes the form of workflows containing action prerequisites. Specifically, WISE-Flow induces **workflows together with prerequisite-augmented action blocks**, carrying both a task-level procedural backbone and the conditions under which actions tend to be valid. During the deployment, WISE-Flow **aligns the ongoing interaction trace to the**

[†]The research was done during an internship at Amazon.

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026, pages 3124–3143
November 6-10, 2026 ©2026 Association for Computational Linguistics

retrieved workflows and performs prerequisite-aware feasibility checks, turning retrieved experience into stage-appropriate next-step guidance that is grounded in the agent's current world state, rather than leaving the model to infer the procedure and feasibility from scratch.

In addition to the framework, we refine agent evaluation. Prior work emphasizes end-to-end success (e.g., task success rate) (Barres et al., 2025; Lu et al., 2025) but overlooks procedural action errors that also affect user experience. We adapt F_β (Chinchor, 1992) to service-agent evaluation, jointly measuring milestone progress and error control beyond final success.

In summary, our contributions are as follows: (1) We propose a three-phase framework, workflow-induced structured experience (WISE-Flow), that automatically induces workflows and prerequisite-augmented action blocks from raw interaction logs and integrates them into a generic tool-augmented agent via workflow-level and action-level retrieval. (2) We introduce a new workflow representation and an evaluation metric F_β to jointly reflect milestone completion and error control. (3) We demonstrate the gains of WISE-Flow across benchmarks and base models. Specifically, it improves F_β by at least 3% on τ^2-bench over the strongest baseline and achieves the highest success rate.

2 Related Work

Tool-augmented conversational agents and benchmarks. Tool-augmented conversational agents extend LLMs by enabling them to interact with external tools and environments, providing access to external information and executable capabilities beyond the models' static knowledge (Lewis et al., 2020; Schick et al., 2023; Qin et al., 2023; Mialon et al., 2023). They often follow an iterative action–observation loop, where tool calls produce environment feedback that guides subsequent decisions (Nakano et al., 2021; Yao et al., 2022). Building on this paradigm, prior work has studied both tool orchestration and improving tool invocation reliability through training and large-scale tool-use supervision (Patil et al., 2024; Schick et al., 2023). In user-facing services, tool use is typically stateful and procedural. Later actions depend on intermediate results, backend state, and coordination with the user (Yao et al., 2024; Lu et al., 2025; Barres et al., 2025). Accordingly, recent benchmarks increasingly evaluate agents under evolving world states and long-horizon interactions, such as τ-bench, ToolSandbox, and τ^2-bench (Yao et al., 2024; Lu et al., 2025; Barres et al., 2025). These settings motivate our focus on execution-time guidance, i.e., reusing past interaction experience as procedural knowledge to guide the next action under the current world state.

Self-evolving agents and memory-augmented experience reuse. To improve reliability in long-running deployments, an emerging direction is to build self-evolving agents that improve over time from past interaction experience (Gao et al., 2025; Wang et al., 2023). A common approach is to equip agents with explicit memory, storing past interactions and retrieving them to inform future decisions (Gao et al., 2025; Zhang et al., 2025; Zhao et al., 2024; Park et al., 2023; Chhikara et al., 2025; Salama et al., 2025). In conversational settings, some methods maintain high-level summaries or reflective memories that capture general takeaways from past dialogues (Zhong et al., 2024; Shinn et al., 2023). Other methods store step-level trace memories, such as tool calls, events, or state-action records, enabling retrieval of locally relevant steps (Zhang et al., 2023; Xu et al., 2025). While these memory forms support experience reuse, they often leave open how to represent and apply past experience as *procedural* guidance during execution in stateful tool environments, especially when the agent must localize progress and select a feasible next action under the current world state.

Procedural experience. Recent work seeks to turn interaction trajectories into reusable procedural knowledge for tool execution. A first line produces textual procedural guidance, distilling experience into artifacts such as conditional guidelines (Fu et al., 2024), concise retrievable hints (Nekoei et al., 2025), or manual-like instructions (Chen et al., 2024) that can be consulted during planning. These memories are easy to store and reuse, but they are usually not tied to the agent's exact stage in an ongoing execution and often leave the applicability of actions implicit. A second direction focuses on structured procedural representations (Wang et al., 2025b; Fang et al., 2025). While such representations capture stepwise procedural knowledge, they are typically retrieved and injected as the task-level context, without explicitly aligning the current interaction trace to a procedure step or validating next-step feasibility at execution time.

An extended discussion about related work is

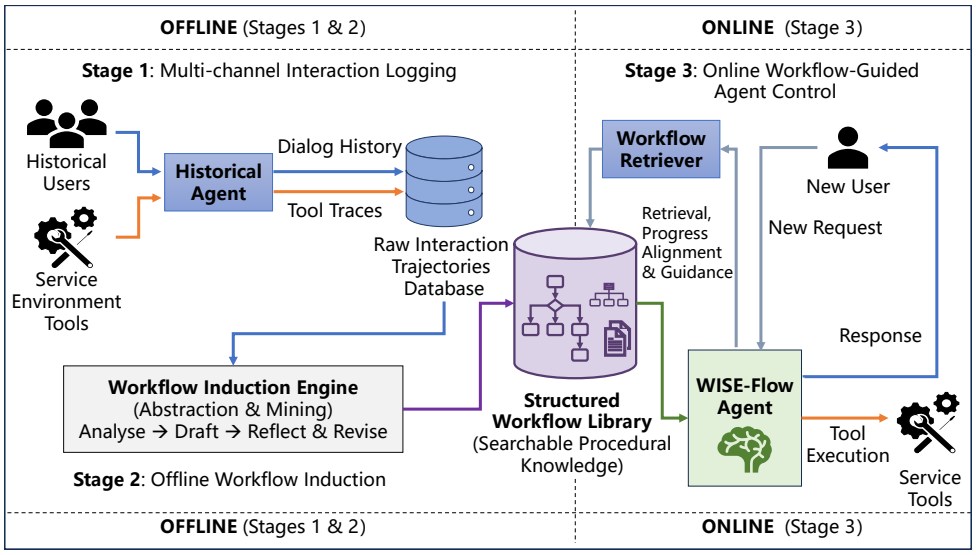


Figure 1: WISE-Flow pipeline.

provided in Appendix A.1.

3 Methodology

In this section, we introduce WISE-Flow, a pipeline for tool-augmented conversational agents operating in task-oriented service environments. The agent's goal is to resolve the user's request as efficiently as possible, with minimal steps and errors. To achieve the objective, we treat historical interaction trajectories as reusable procedural knowledge. Specifically, we extract structured workflows from past interactions and organize them into a searchable library. At inference time, the agent retrieves workflows relevant to the new task and uses them as actionable guidance and soft constraints when planning its next actions.

At a high level, WISE-Flow follows a three-stage pipeline as shown in Figure 1: (1) collect full logs from multiple sources, including dialog, tool traces and environment feedback, (2) automatically induce structured workflows from the collected logs, and (3) guide agents using the induced workflows. Next, we describe each stage in turn.

3.1 Multi-channel Interaction Logging

The first stage builds a repository of service logs. In task-oriented services, many failures are procedural rather than linguistic, such as executing actions in an invalid order or violating implicit prerequisites. Such failures are often only revealed through environment-side feedback, and are difficult to diagnose from dialog text alone. Consequently, dialog-only summaries are insufficient to recover *reusable procedures*. Unlike previous work (Choubey et al., 2025), WISE-Flow explicitly emphasizes the importance of modeling the *complete service log*, not just a sequence of utterances, but a sequence of events from multiple sources. The richer record can provide the evidence needed to recover reliable procedures offline for future reference. Formally, each trajectory is modeled as an ordered event stream: $\tau = \{e_t\}_{t=1}^{T}$, $e_t = (r_t, x_t)$, where $r_t \in \{user, assistant, environment\}$ means the source of the event, and x_t is its content, which can be user messages, assistant responses, tool invocations, tool outputs, and backend signals such as API return codes and error messages.

3.2 Offline Workflow Induction

Given a set of service trajectories $\mathcal{T}$ collected for a task, the second stage of WISE-Flow performs offline workflow induction, distilling messy, execution-level logs into *reusable and actionable procedures*. Unlike prior work that summarizes successful traces into narrative guidance or local rules, WISE-Flow induces a *structured workflow with explicit, environment-grounded prerequisites*, leveraging *contrastive* evidence from clean successes, error-recovered successes, and failures. Even with complete service logs, raw trajectories are often not directly reusable as procedures. They contain detours and errors, and different trajectories may succeed via different paths. The key challenge is to recover a *stable* procedure with *executable constraints* such as prerequisites and ordering. WISE-Flow turns outcome-divergent traces

into supervision, where contrastive comparisons isolate the minimal deviations that separate success from failure and encode them as explicit ordering constraints and prerequisites in the workflow.

Pre-processing and contrast construction. We partition trajectories for each task into three categories: *clean successes* (no tool-call errors), *recovered successes* (contain tool-call errors but later recover), and *failures*. We then construct paired contrasts by pairing a clean success with a recovered success or a failure. Concatenating each pair forms a contrastive block $\mathcal{C}(\mathcal{T})$, which makes the outcome-determining differences explicit. The block highlights **effective action ordering** in successes and reveals violated **prerequisites and ordering constraints** via environment feedback in the recovered or failed traces. Meanwhile, we use an LLM to produce a task summary $s(\mathcal{T})$ from the user queries, as shown in Appendix Figure 8.

Workflow induction. WISE-Flow induces a workflow via $w = g(s(\mathcal{T}), \mathcal{C}(\mathcal{T}))$, where $g(\cdot)$ is an LLM-based workflow inducer that takes the task summary $s(\mathcal{T})$ and the contrastive block $\mathcal{C}(\mathcal{T})$ as input, and outputs a workflow w under a constrained JSON schema (Appendix Fig. 6d). Intuitively, $s(\mathcal{T})$ describes the global task intent, while $\mathcal{C}(\mathcal{T})$ provides discriminative supervision about what mattered for success, which is inspired by AutoGuide (Fu et al., 2024) but targets a richer procedural representation than local decision rules. Directly generating procedures from logs is brittle, as LLMs may hallucinate steps, omit execution-critical constraints, or make ungrounded recommendations. Therefore, we propose the **grounded induction via multi-pass verification**, which separates evidence extraction from synthesis and verifies support in the input trajectories. Specifically, we use a **three-pass procedure** (Appendix Fig. 6): (i) **analysis** identifies the goal-consistent action sequence and the key environment feedback explaining failures; (ii) **drafting** synthesizes a candidate workflow with explicit steps and per-action prerequisites; and (iii) **reflection and revision** validates each step, prerequisite, and branching condition against trajectories, removing or fixing unsupported or non-executable recommendations. This verification-based design improves faithfulness and executability over single-pass generation. For existing task types, the same analysis–drafting–revision process can update the current workflow with new logs, refining prerequisites and recovery paths while pruning obsolete trial-and-error steps instead of storing each trajectory separately.

Workflow representation. To bridge the retrieval-to-action gap, we design a workflow representation that couples a *task-level workflow* with *prerequisite-augmented action blocks* (see Appendix Fig. 6d and Fig. 9). The workflow backbone summarizes the task with a short description and ordered milestones, which support progress alignment. Each action block specifies an action with explicit prerequisites (both global and scenario-specific) and conditional next-step transitions grounded in environment feedback. Prerequisites act as execution guards against common procedural mistakes, and transitions encode branching behavior observed in past trajectories. As a result, retrieved experience becomes actionable guidance with explicit success/recovery routes and checkable prerequisites, enabling feasibility-aware next-step selection.

3.3 Online Workflow-guided Agent Control

In the third stage, WISE-Flow assists conversational agents during deployment with the induced workflow library (see the prompt in Appendix Figure 7). Retrieving past experience is not enough. The challenge is that the agent must (i) localize its current stage within a multi-step procedure and (ii) select a next action that is feasible under the current world state. Without explicit progress alignment and prerequisite checking, retrieved experience provides limited support for a feasible next action. WISE-Flow therefore converts retrieved workflows into actionable next-step guidance via **progress alignment and prerequisite-aware feasibility checking**. Let h_t denote the current interaction history up to turn t: $h_t = (e_1, e_2, ..., e_t)$. Then, we extract from h_t a summary of the user's request as the task-level query q_t via an LLM summarizer, and identify the current tool trace as well as the last successfully executed tool a_{t-1}.

Workflow retrieval. Each induced workflow w contains a task description, entry steps, planned steps, and action blocks, and is indexed by an embedding $\mathbf{v}_w$. We embed the task query as $\mathbf{u}_t = \phi(q_t)$ and retrieve top-K workflows by $\mathcal{W}_t = TopK_{w \in \mathcal{W}} < norm(\mathbf{u}_t), norm(\mathbf{v}_w) >$, where $\phi(\cdot)$ is the embedding model and $\mathcal{W}$ is the workflow library.

Progress alignment. Retrieval provides relevant workflows but does not locate the agent within a procedure. We therefore align the last successful tool call a_{t-1} to the planned-step list $S^{(w)}$, select the most similar step, and take its successor as the

next-step candidate. This converts retrieved workflows into stage-specific suggestions and narrows the next-action space to a small set of task-relevant and procedurally coherent actions.

Prerequisite-aware action validation. Stage-conditioned suggestions may be infeasible due to unmet prerequisites. Using prerequisite annotations in retrieved action blocks, we check each candidate action against the interaction trace and environment feedback observed so far (see the prompt in Appendix Figure 8). Then, we return candidates with prerequisite status (met or unmet), guiding the agent to either execute the action directly or resolve missing prerequisites first. This makes feasibility explicit, instead of requiring the model to infer constraints from scratch.

Workflow-conditioned context injection. Finally, we construct a compact structured context block, which contains the retrieved workflows, suggested next action candidates, and prerequisite status for each candidate. Compared with conventional context injection, it provides actionable next-step guidance with checkable constraints, enabling more reliable tool execution under the current world state.

4 Experimental Setup

We evaluate the performance of WISE-Flow through several controlled experiments, focusing on: **(RQ1)** End-to-end performance gains of WISE-Flow. **(RQ2)** The impact of modeling complete service logs. **(RQ3)** The effect of structured experience representation. **(RQ4)** The benefit of aggregating trajectories per task for workflow induction.

4.1 Benchmarks and Environments

We use two controllable simulation environments to collect interaction trajectories as past experience for offline workflow induction, and we evaluate our method in the same environments.

ToolSandbox (Lu et al., 2025) provides an executable, stateful service environment for multi-turn tool use, where tool invocations induce persistent state transitions and produce fine-grained feedback. Intermediate environment states are observable. An agent's success in ToolSandbox depends on executing the correct tool calls in the right order. These properties make it well-suited for our setting. Our evaluation is done on 76 tasks of ToolSandbox spanning both multiple-tool-call scenarios and multiple-user-turn scenarios.

τ^2-bench (Barres et al., 2025) evaluates conversational agents in a dual-control setting, where both the agent and the user can invoke tools to update a shared world state. It targets collaborative troubleshooting that demands coordination and grounded communication, while exposing procedural failures and recovery cues via environment-side feedback. In our experiments, we evaluate agents on the telecom domain, the most complex domain of the τ^2-bench, which contains 114 tasks.

4.2 Baselines

We compare WISE-Flow with six representative strategies for agent design (see Appendix A.2.2).

LLM-only uses the base tool-augmented agent without additional memory, reflection, or workflow guidance. It relies entirely on the backbone LLM's implicit reasoning and instruction-following capabilities to decide each response or tool action.

ReAct (Yao et al., 2022) structures agent decision making as an interleaved reasoning–acting loop. The agent alternates between natural language reasoning and tool invocations, using tool observations to revise its intermediate plans. ReAct improves within-episode control through immediate feedback, but does not provide an explicit mechanism for cross-episode experience reuse. In our implementation, we prompt the agent to follow the standard Thought-Action-Observation format.

Reflexion (Shinn et al., 2023) equips an agent with post-hoc self-reflection. After each episode, an LLM analyzes the trajectory, identifies failure modes, and summarizes lessons as textual memories. For a new task, the agent retrieves relevant reflections and conditions its reasoning on them, aiming to avoid repeating past mistakes and improve cross-episode performance. In our implementation, we follow this protocol by generating reflection summaries from historical service logs and injecting the retrieved reflections into the agent context at test time.

REMEMBERER (Zhang et al., 2023) constructs an experience memory from past trajectories. REMEMBERER maintains a persistent tabular memory, where each record contains a task goal, an observation, an action taken under that observation, and a value estimate (e.g., a Q-value) for that action. At inference time, given the current task and observation, the agent retrieves the most similar records and uses their value estimates to rank and bias action selection, preferring high-value actions while discouraging lower-value alternatives among the retrieved candidates.

AutoGuide (Fu et al., 2024) distills offline agent trajectories into concise, context-aware guidelines expressed as conditional rules, where the context is the summary of the current status. At test time, the agent retrieves and follows these guidelines to bias action choices under similar contexts.

AWM (Wang et al., 2025b) was proposed for web-navigation agents, where it induces reusable workflows from successful agent trajectories and retrieves them as workflow-level memory, without contrastive failure/recovery evidence or execution-time prerequisite checking.

4.3 Models

We use Claude 3.7 Sonnet (Anthropic, 2025) and Qwen3-235B-A22B (Qwen-Team, 2025) as the base models for our agents, and use Claude 3.7 Sonnet for user simulation. We focus on deployable frontier service agents, where the base model should already reliably follow tool syntax and track multi-turn state. Evaluating whether the same guidance benefits smaller models is left to future work. We set the temperature to 0.9 when collecting diverse experience, and to 0 during testing. More details are in Appendix A.2.

4.4 Evaluation Metric

We evaluate the agent on both benchmarks using three metrics: (1) the average task success rate, (2) the average missed-milestone ratio, and (3) the F_β score. Let N be the number of task. The average success rate is $SR = \frac{1}{N}\sum_{i=1}^{N} I_{[\textit{task}_i \textit{ is solved}]}$, where I is the indicator function. The average missed-milestone ratio is $MMR = \frac{1}{N}\sum_{i=1}^{N} \frac{m_i^{miss}}{m_i}$, where m_i denotes the required milestones for task i and m_i^{miss} denotes those not achieved.

Most evaluations of conversational agents emphasize end-to-end success (Barres et al., 2025; Lu et al., 2025), such as the task success rate, but often ignore action errors during execution. In user-facing services, reliability (i.e., steady progress without failures) matters as much as the final outcome. Therefore, we introduce F_β (Chinchor, 1992) into the evaluation of tool-augmented conversational agents to jointly capture milestone coverage and execution reliability. Similar to F_1, F_β is a weighted harmonic mean of milestone recall R and an error-aware precision term P:

$$F_\beta = \frac{(1+\beta^2)\, P * R}{\beta^2 P + R}, \tag{1}$$

where β is a metric hyperparameter that adjusts the relative importance of recall versus precision. $\beta > 1$ emphasizes recall. In the following results, we report F_β with $\beta = 5$, since failing to cover required milestones is more harmful than making occasional errors. We define

$$R = \frac{m_i - m_i^{miss}}{m_i},\ P = \frac{|A_i| - |E_i|}{|A_i|}. \tag{2}$$

where $|A_i|$ denotes the total number of actions made by the agent in task i, and $|E_i|$ denotes the number of action errors encountered in task i.

To better understand how the agent succeeds, we further break down successful trials into **one-shot successes** (solved without trial-and-error) versus **successes after iterative correction**. We quantify the reliance on trial-and-error by computing the proportion of trial-and-error successes among all successful trials: $\text{TE-ratio} = \frac{\text{\# (success after trial-and-error)}}{\text{\# (all successful trials)}}$.

Besides these metrics, we also report the default metrics provided by each benchmark, i.e., the similarity score of ToolSandbox, which measures the similarity between the agent's actions and resulting world states and the ground truth, and $pass\hat{}k$ of τ^2-bench (Yao et al., 2024), which defines the probability that an agent succeeds on the same task for k i.i.d. trials (i.e., all k runs are successful), averaged across tasks.

5 Results and Analysis

5.1 Overall Results (RQ1)

We simulate 10 independent trials to collect the raw experience, i.e., the initial trajectories, for each task. During evaluation, the agent retrieves the top K most similar experiences to guide its actions. We set $K = 3$, run each task 10 times on ToolSandbox and 3 times on the τ^2-bench, and report the performance averaged over all tasks and trials. The overall results are shown in Table 1 and Table 2. For τ^2-bench, since the $pass\hat{}1$ is equivalent to the success rate, we omit the success rate in Table 2.

WISE-Flow improves success while controlling procedural errors. On ToolSandbox, WISE-Flow achieves the best overall performance, obtaining the highest success rate and the lowest MMR.

WISE-Flow reduces trial-and-error during execution. Figure 2 shows that WISE-Flow has a lower TE-ratio than REMEMBERER, indicating that successful runs rely less on trial-and-error.

Model	Method	Similarity ↑	F_β ↑	MMR ↓	SR ↑
Qwen3-235B-A22B	LLM-only	0.856	0.8985	9.75%	77%
	ReAct	0.858	0.9008	9.70%	76%
	Reflexion	0.866	0.8999	9.79%	84%
	REMEMBERER	0.871	0.9134	8.51%	78%
	AutoGuide	0.911*	0.9484*	5.08%*	87%*
	AWM	0.853	0.889	10.91%*	75%*
	WISE-Flow (Ours)	**0.923**	**0.9615**	**3.69%**	**88%**
Claude 3.7 Sonnet	LLM-only	0.871	0.9277	6.99%	87%
	ReAct	0.911*	0.9603	3.72%	92%
	Reflexion	0.875	0.9462	5.42%	87%
	REMEMBERER	**0.914**	0.9728*	2.47%*	**95%**
	AutoGuide	0.911*	0.9699	2.90%	91%
	AWM	0.862	0.926	7.21%	87%
	WISE-Flow (Ours)	0.910	**0.9745**	**2.46%**	95%*

Table 1: Results on ToolSandbox. Best results are in bold and second-best are with * . Same for all tables. Standard deviations are reported in Appendix Table 5.

Model	Method	pass^1 ↑	pass^2 ↑	pass^3 ↑	F_β ↑	MMR ↓
Qwen3-235B-A22B	LLM-only	0.395*	0.222	0.132	0.5537	43.84%
	ReAct	0.348	0.222*	0.184*	0.4217	57.50%
	Reflexion	0.345	0.216	0.167	0.5456	45.66%
	REMEMBERER	0.319	0.164	0.097	0.4792	52.05%
	AutoGuide	0.322	0.211	0.149	0.4907	51.12%
	AWM	**0.418**	0.249*	0.158	0.6015*	39.23%*
	WISE-Flow (Ours)	0.395*	**0.252**	**0.193**	**0.6343**	**36.65%**
Claude 3.7 Sonnet	LLM-only	0.462	0.339	0.272	0.6329	36.92%
	ReAct	0.506	0.359	0.278	0.6357	36.59%
	Reflexion	0.535*	0.433*	0.377*	0.6540	34.65%
	REMEMBERER	0.523	0.424	0.368	0.6629*	33.92%*
	AutoGuide	0.453	0.366	0.333	0.6067	39.62%
	AWM	0.471	0.345	0.281	0.6183	38.28%
	WISE-Flow (Ours)	**0.564**	**0.468**	**0.421**	**0.6936**	**30.75%**

Table 2: Results on τ^2-bench. Standard deviations are reported in Appendix Table 6.

Reflexion also has a low TE-ratio, possibly because failure reflections help avoid repeated mistakes. However, Reflexion lacks the global planning mechanism used in WISE-Flow, which limits its ability to solve complex tasks end-to-end and results in a lower overall success rate. Overall, **WISE-Flow achieves the best balance between high success and low error rate**, which is also consistently reflected by its F_β score. This also shows F_β **provides a more holistic and discriminative evaluation of agent performance than any single metric alone.**

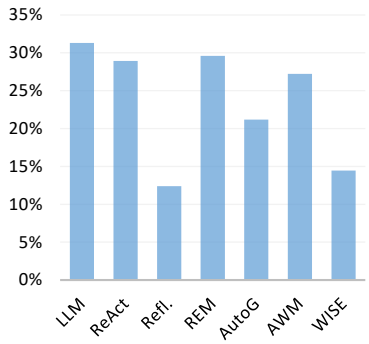


Figure 2: Proportion of trial-and-error (TE) successes among all successful trials (lower is better). We decompose successful episodes into one-shot successes and successes after TE, and report the proportion of the latter for each method. (Refl. = Reflexion, REM = REMEMBERER, AutoG = AutoGuide, WISE= WISE-Flow)

Stage-specific and feasibility-aware guidance improves dynamic task execution. On τ^2-bench, WISE-Flow achieves the best results with Claude 3.7 Sonnet across $pass\hat{}k$ for $k = 1, 2, 3$, F_β, and missed-milestone ratio. With Qwen3-235B-A22B, all baselines except AWM have lower $pass\hat{}1$ than

the LLM-only agent, but degrade more slowly as k increases and can surpass it under stricter consistency requirements. **This weaker single-run performance suggests a guidance-mismatch effect for memory-based baselines in stateful conversational services**. A detailed analysis of their failure modes is provided in Appendix A.3.1. AWM achieves the highest $pass\hat{}1$, but its $pass\hat{}k$ drops more quickly as k increases. This suggests that directly reusing past successful workflows can help in single-run settings, but implicit action applicability and prerequisites are less stable under varied multi-turn interactions. In contrast, **WISE-Flow uses contrastive workflow induction and grounds suggested actions in explicit prerequisites and the current service state, making it less reliant on copying a prior successful trajectory and more robust** as the $pass\hat{}k$ consistency requirement increases. WISE-Flow also attains the highest F_β among all methods, indicating **a better balance between milestone coverage and error control**.

We also evaluate a leave-one-task-out variant of WISE-Flow that excludes same-task experience. As shown in Appendix A.3.3, WISE-Flow remains competitive and outperforms several memory baselines, suggesting that it can **generalize across tasks** by capturing reusable procedural patterns beyond task-specific trajectory memorization.

Overall, WISE-Flow achieves consistently strong performance across different backbones and environments.

5.2 Effectiveness of Backend Signals (RQ2)

Next, we investigate the effect of using full logs as past experience, which capture not only user-agent interactions but also the agent's actions and environment feedback, such as error messages.

We use WISE-Flow to extract two different workflows: one from full logs and the other from dialogue-only traces without environment feedback. We then apply these two workflow variants to the agents and compare their performance. We use ground-truth-based retrieval, meaning that the retrieved workflow comes from the same task instance, thereby eliminating retrieval randomness. We conduct this experiment on ToolSandbox, running each task for 10 trials.

From Table 3, we can observe a clear performance gap between the two variants. Using full logs improves milestone coverage by around 3% and raises F_β by 3%, showing that environment feedback improves both milestone completion and execution reliability. We further observe that the overall success rate increases by 3.9%, while the proportion of trials that succeed only after trial-and-error decreases by 3%, suggesting that the improvement is not only in final task completion, but also in one-shot success. This further demonstrates that the gain in F_β is driven not merely by a higher SR, but also by reduced errors during execution.

Overall, these results demonstrate that **environment feedback is essential when using past experience for agent guidance**. The trial-and-error signals embedded in tool outputs and error messages provide valuable supervision, enabling agents to complete tasks more effectively while requiring fewer corrective attempts.

	FT	Dialogue-Only	Δ
Similarity ↑	**0.9136**	0.8852	↓ 2.8%
F_β ↑	**0.9755**	0.9457	↓ 3.0%
MMR ↓	**2.5%**	5.4%	↑ 3.0%
SR ↑	**93.5%**	89.6%	↓ 3.9%
SR with trial-and-error ↓	**8.5%**	11.5%	↑ 3.0%

Table 3: Ablation on log completeness: workflows extracted from full traces (FT) vs. dialogue-only traces. Δ denotes the per-row difference between the two settings.

5.3 Benefits of Structured Experience Representations (RQ3)

We next study how experience structure affects agent guidance. We compare three representations: (i) raw service logs, (ii) a plain-text workflow describing the action order, and (iii) the structured WISE-Flow representation. We evaluate on ToolSandbox with Claude 3.7 Sonnet and use ground-truth-based retrieval, where the retrieved experience comes from the same task, to remove retrieval randomness and isolate the effect of representation.

From Table 4, we observe that experience representation plays a critical role in downstream performance. Both Raw Logs and Plain-text Workflow outperform the LLM-only baseline across all metrics, indicating that leveraging past experience is beneficial. Among them, WISE-Flow achieves the best results overall, with the highest similarity, SR, and F_β, as well as the lowest MMR. Notably, while the plain-text workflow description improves final task success, it does not improve one-shot success over raw conversations, suggesting that unstructured or loosely structured experience may still lead to trial-and-error execution. In contrast, WISE-Flow substantially increases one-shot suc-

Metric	LLM	Raw Logs	Plain WF	WISE
Similarity ↑	0.8707	0.8879	0.8989	**0.9136**
F_β ↑	0.9277	0.9449	0.9584	**0.9755**
MMR ↓	6.99%	5.45%	4.06%	**2.46%**
SR ↑	86.97%	88.88%	93.00%	**93.50%**
One-shot SR ↑	59.74%	71.63%	70.88%	**85.00%**

Table 4: Effect of experience representation on ToolSandbox. Plain WF=Plain-text Workflow.

cess, demonstrating that **structured experience not only helps agents complete tasks but also reduces corrective attempts by better guiding action selection and error avoidance**.

5.4 Benefits of Aggregating Trajectories for Workflow Construction (RQ4)

We then show that aggregating multiple trajectories from the same task yields better workflow guidance. Given several trajectories for a task, we compare two WISE-Flow induction strategies: (i) **Trajectory-wise** extraction, which constructs one workflow from each trajectory, and (ii) **Task-wise** extraction, which aggregates same-task trajectories into a single consolidated workflow. We evaluate both on ToolSandbox under the same retrieval method and setup.

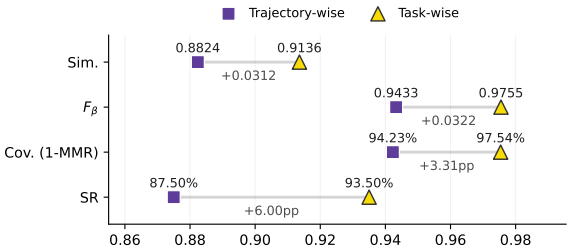


Figure 3: Trajectory-wise vs. task-wise workflow induction on ToolSandbox. Higher Value is better. (We report milestone coverage as $1 - \mathrm{MMR}$. Sim. = Similarity.)

The results show a clear advantage of task-wise workflow induction over trajectory-wise induction. As shown in Figure 3, aggregating multiple trajectories from the same task yields consistent improvements across all metrics, leading to about a 3% gain in similarity, F_β, and milestone coverage, as well as a 6% gain in success rate. We believe this improvement mainly comes from comparing and aggregating multiple attempts during workflow induction. A workflow induced from a single trajectory can include unnecessary or false trial steps, whereas task-wise induction lets the LLM compare different attempts and summarize a more stable, task-relevant action pattern into one consolidated workflow. This is more effective than asking the agent to compare all trajectories during task execution, leading to clearer guidance, fewer mistakes, and higher performance.

Besides the observations above, our supplementary analysis also demonstrates that: (i) the proposed three-pass procedure outperforms single-shot induction (Appendix A.3.2); (ii) WISE-Flow generalizes across tasks, enabling performance gains when retrieving workflows from other tasks (Appendix A.3.3); and (iii) WISE-Flow retrieves genuinely useful workflows and benefits from them (Appendix A.3.4). We also include a hyperparameter study in Appendix A.3.5.

6 Conclusion

In this paper, we introduce WISE-Flow, a framework that turns raw service logs into structured, reusable workflows and guides tool-augmented conversational agents via retrieval at both the workflow and action levels. We also propose a new workflow representation and introduce a new evaluation metric F_β. Experiments on ToolSandbox and τ^2-bench show that WISE-Flow improves performance and better balances success and errors, as reflected by higher F_β. Our analyses indicate the importance of environment feedback and show that structured, trajectory-aggregated experience provides stronger guidance and can transfer across tasks, even without the same-task experience. Overall, WISE-Flow makes past tool-use experience more usable and effective for LLM-based agents in conversational service environments.

Limitations

Despite its effectiveness, WISE-Flow has several limitations.

Dependence on log observability. WISE-Flow assumes access to complete service trajectories that include not only dialog turns but also tool traces and environment-side feedback. In many real deployment, such information may be partially missing. When the environment-side feedback is incomplete, the induced workflow, along with action prerequisites and procedural constraints, becomes harder to recover and may not fully capture the successful strategies, which may reduce the benefit of workflow guidance.

Reliability of offline workflow induction. The

workflow induction of WISE-Flow relies on an LLM to synthesize structured procedures from trajectories. Although we employ a multi-pass induction procedure to reduce unsupported steps, the inducer may still introduce spurious ordering constraints, include unnecessary steps, or hallucinate prerequisites that appear plausible but are not strictly necessary, which can be harmful in online deployment, especially if the agent treats them as strong guidance. Incorporating executable verification or learning minimal sufficient procedures remains an important direction.

Progress localization and branching complexity. The online control of WISE-Flow localizes agent progress by aligning recent successful tool calls to the planned steps of retrieved workflows and proposing stage-specific next actions. This heuristic can be brittle in workflows with substantial branching, repeated actions across stages, or non-sequential interleavings of tool calls. In such cases, matching only a short tool trace may misidentify the current stage and narrow the action shortlist incorrectly. Extending progress alignment to sequence-level matching, state-aware alignment, or probabilistic stage tracking may improve robustness for more complex tasks.

Ethical Considerations

Our research complies with applicable ethical standards and relies exclusively on publicly available datasets, benchmarks, and environments. All simulations are performed strictly within the scope, scenarios, and constraints specified by these resources. The contents of the datasets, benchmarks, and environments are provided by their respective sources and do not reflect the views, opinions, or positions of the authors.

Potential Risks and Mitigations. Our work is evaluated using ToolSandbox and τ^2-bench, which are controlled environments and benchmarks. While these resources are intended for safe and reproducible research, the proposed WISE-Flow framework can introduce dual-use and unintended harmful effects if deployed outside controlled settings. First, the ability to induce and guide multi-step tool usage workflows could be repurposed to automate undesirable or abusive actions, e.g., large-scale API abuse, unauthorized automation of workflows in sensitive systems, even if the original evaluation is in simulation. Second, errors in workflow induction or progress localization may lead an agent to issue inappropriate tool calls. In real environments connected to external services, such guidance errors could cause irreversible effects, waste resources, or trigger unsafe behaviors.

We mitigate these potential harms by (i) emphasizing that our evaluation is limited to simulated sandbox environments and not intended for direct deployment in high-stakes real-world systems, (ii) recommending restrictions and confirmation steps for high-impact or irreversible tools, (iii) sanitizing or abstracting sensitive fields in task traces used for workflow induction, and (iv) encouraging future work to incorporate executable verification, safety filters, and fairness evaluations across diverse domains. These safeguards help ensure that workflow guidance enhances benign automation without exacerbating misuse, privacy violations, or unfair outcomes across stakeholder groups.

References

Anthropic. 2025. Claude 3.7 sonnet system card. https://www.anthropic.com/claude-3-7-sonnet-system-card. System card for Claude 3.7 Sonnet.

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*.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. 2024. Automanual: Constructing instruction manuals by llm agents via interactive environmental learning. *Advances in Neural Information Processing Systems*, 37:589–631.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.

Nancy A. Chinchor. 1992. Muc-4 evaluation metrics. In *Message Understanding Conference*.

Prafulla Kumar Choubey, Xiangyu Peng, Shilpa Bhagavath, Caiming Xiong, Shiva Kumar Pentyala, and Chien-Sheng Wu. 2025. Turning conversations into workflows: A framework to extract and evaluate dialog workflows for service AI agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 3933–3954, Vienna, Austria. Association for Computational Linguistics.

Cohere. 2023. Introducing embed v3. https://cohere.com/blog/introducing-embed-v3.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou.

2025. The faiss library. *IEEE Transactions on Big Data*.

Shengda Fan, Xin Cong, Zhong Zhang, Yuepeng Fu, Yesai Wu, Hao Wang, Xinyu Zhang, Enrui Hu, and Yankai Lin. 2026. Generalizing experience for language agents with hierarchical metaflows. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

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

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Advances in Neural Information Processing Systems*, 37:119919–119948.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, and 1 others. 2025. A survey of self-evolving agents: On path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

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, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, and 1 others. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Hadi Nekoei, Aman Jaiswal, Patrice Bechard, Oleh Shliazhko, Orlando Marquez Ayala, Mathieu Reymond, Massimo Caccia, Alexandre Drouin, Sarath Chandar, and Alexandre Lacoste. 2025. Just-in-time episodic feedback hinter: Leveraging offline knowledge to improve llm agents adaptation. *arXiv preprint arXiv:2510.04373*.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Qwen-Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Rana Salama, Jason Cai, Michelle Yuan, Anna Currey, Monica Sunkara, Yi Zhang, and Yassine Benajiba. 2025. Meminsight: Autonomous memory augmentation for llm agents. *arXiv preprint arXiv:2503.21760*.

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.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

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*.

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

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. 2025a. Inducing programmatic skills for agentic tasks. In *Second Conference on Language Modeling*.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025b. Agent workflow memory. In *Forty-second International Conference on Machine Learning*.

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*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ^2-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. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Danyang Zhang, Lu Chen, Situo Zhang, Hongshen Xu, Zihan Zhao, and Kai Yu. 2023. Large language models are semi-parametric reinforcement learning agents. *Advances in Neural Information Processing Systems*, 36:78227–78239.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A Appendix

A.1 Related Work

AWM (Wang et al., 2025b) is closest to our work, which induces workflows from agent trajectories for web navigation and reuses them as workflow-level memory. However, AWM primarily injects retrieved workflows as procedural demonstrations, keeps action applicability implicit in step descriptions, and does not explicitly validate whether a retrieved action is feasible under the current tool state. Different from AWM, WISE-Flow targets stateful conversational service agents, where failures are often exposed through backend tool outputs and environment feedback. It induces workflows from complete service logs with contrastive evidence from clean successes, recovered successes, and failures, augments workflows with explicit action prerequisites, and uses progress alignment plus prerequisite-aware feasibility checks to turn retrieved experience into stage-appropriate next-step guidance.

MetaFlowLLM (Fan et al., 2026) also studies structured experience reuse. It constructs a hierarchical experience tree with reinforcement learning using verifiable correctness and efficiency rewards, in which each MetaFlow contains static workflow steps and dynamic subtasks. This is complementary to WISE-Flow. MetaFlowLLM optimizes reusable workflows in reward-verifiable task environments, whereas WISE-Flow focuses on conversational service settings where intermediate rewards can be sparse, expensive, or unavailable and where action feasibility depends on user turns and backend state. Rather than training a workflow generator with RL, WISE-Flow extracts environment-grounded prerequisites from historical service logs and checks them during execution.

ASI (Wang et al., 2025a) is related but more distant. It induces executable programmatic skills for web tasks, thereby expanding the agent's action space. WISE-Flow instead keeps the original service-tool interface and provides workflow-level guidance over dialogue, tool-use traces, and backend feedback.

More broadly, most procedural-reuse methods are studied in settings that tolerate exploratory trial-and-error during execution, such as web navigation (Zhou et al., 2023; Wang et al., 2025a) and embodied interactive simulators (Shridhar et al., 2020). In contrast, we focus on user-facing services, where procedural failures directly degrade user experience. Therefore, agents should minimize intermediate errors, not just achieve end-to-end success.

A.2 Experimental Setup

A.2.1 Historical Experience Simulation

When simulating the historical experiences, we set the temperature as 0.9. For user simulation, we use Claude 3.7 Sonnet (Anthropic, 2025). For agent simulation, we use Claude 3.7 Sonnet ("us.anthropic.claude-3-7-sonnet-20250219-v1:0") and Qwen3-235B-A22B (Qwen-Team, 2025) ("qwen.qwen3-235b-a22b-2507-v1:0") as the base models. We run 10 trials per benchmark per agent using the environment's default prompts.

A.2.2 Implementation of Baselines

All methods (including ours) are treated equally in the prompt tuning budget. All baselines are optimized using the same automated LLM-based meta-instruction procedure, reducing human bias and ensuring a fair comparison. The prompts of the baselines in ToolSandbox are shown in Figure 4.

A.2.3 Implementation of WISE-Flow

Offline Workflow Induction. Figure 6 shows the prompts for workflow induction and the workflow format. The model used for workflow induction is the same as the agent's base model. We show a sample of induced workflow in Figure 9.

Workflow Embedding and Index Construction. We use the Cohere.embed-english-v3 (Cohere, 2023) embedding model ("cohere.embed-english-v3") and build a FAISS (Douze et al., 2025) index for similarity search. Specifically, we embed each induced workflow as a rich textual document that includes its description, actions, and prerequisites. Then we index these embeddings with a cosine-similarity FAISS backend. We also build an action-level index over all action blocks. At inference time, the agent embeds the current task query and retrieves top-K workflows and corresponding actions, then maps retrieved vectors back to structured workflow and actions via stored metadata for workflow-guided control.

Online Workflow-guided Agent Control. For online deployment, we use the same system prompt as Reflexion and REMEMBER on ToolSandbox, as shown in Figure 4, and use the prompt shown in Figure 7 on τ^2-bench. We also introduce an LLM-based pre-processing step for online retrieval. As shown in Figure 8, we prompt the same base model as the agent to (i) generate a task summary from the current dialogue as the retrieval query for workflows and (ii) after progress alignment, check whether the prerequisites of the retrieved action blocks are satisfied.

A.3 Experiment Results

A.3.1 Failure Modes of Memory-Based Baselines

On τ^2-bench, WISE-Flow achieves the best results with Claude 3.7 Sonnet across $pass\hat{}k$ for $k = 1, 2, 3$, F_β, and missed-milestone ratio. With Qwen3-235B-A22B, all baselines except AWM have lower $pass\hat{}1$ than the LLM-only agent, but degrade more slowly as k increases and can surpass it under stricter consistency requirements. **This weaker single-run performance suggests a guidance-mismatch effect for memory-based baselines in stateful conversational services**:

1. AutoGuide retrieves guidelines by matching compact state summaries. In stateful service tasks, these summaries can be under-specified because action feasibility depends on fine-grained dialogue constraints, tool outputs, and backend preconditions.

2. Reflexion provides high-level failure lessons, which can help avoid repeated mistakes but often do not specify the next feasible action. Such abstract advice may over-constrain recovery or lead to ineffective action selection.

3. REMEMBERER's local traces do not provide an explicit task-level workflow, progress alignment, or prerequisite validation.

These memories can therefore be broadly relevant but locally inapplicable under dynamic user behavior and backend state changes.

AWM shows a different pattern. It achieves the highest $pass\hat{}1$, but its $pass\hat{}k$ drops more quickly as k increases. This suggests that directly reusing past successful workflows can help in single-run settings, but implicit action applicability and prerequisites are less stable under varied multi-turn interactions.

A.3.2 Effect of Multi-Pass Workflow Induction

We examine the effect of the workflow induction procedure on representation quality, i.e., whether the multi-pass induction of WISE-Flow produces workflows that are more reliable, where we explicitly separate workflow induction into three stages (analyse, draft, and reflect & revise). We compare it with a one-pass setting, where the model completes all stages with a single instruction.

As shown in Table 7, three-pass induction outperforms one-pass induction across all metrics. While it improves the overall success rate by about 3%, the most striking effect is on one-shot success: the one-shot success rate increases by about 20%, and the trial-and-error success rate (i.e., success achieved only after intermediate failures) drops sharply by 17.5%. Together, these results show that three-pass induction does not merely raise final success, but substantially increases the likelihood of

```
# ReAct
You are an intelligent assistant that solves problems step by step using reasoning and actions.
You follow this format in your responses:
1. <Thought>Explain your current reasoning.</Thought>
2. <Action>Take one of the following actions:
    - Take a specific action using one of the available tools. Always format tool calls correctly based on the tool's
required parameters.
    - Or ask the user for clarification.</Action>
3. <Observation>The result of the action (provided externally by tools or the user).</Observation>

Guidelines:
- Only perform **one Thought and one Action per turn**.
- If you are missing key information, do **not make assumptions**. Instead, use an Action to ask the user for
clarification.
- If a tool is needed, specify the tool and arguments clearly in the Action.
- Do not simulate tool results yourself—wait for an Observation to continue.

Examples:
<Thought>I need to know the weather in {LOCATION}.</Thought>
<Action>get_weather("{LOCATION}")</Action>


# Reflexion/REMEMBERER
You are a reasoning-capable assistant agent that can learn from past experience to solve problems more effectively.
You may refer to the guidelines that incorporates lessons from previous similar situations.

At each time step, You need to generate ONLY one action given the current observation.

Notes:
-  Not all guidelines are useful; some may be irrelevant or misleading.
-  Treat guidelines as soft guidance, not hard rules.
-  Always make the final decision based on the current context.


# AutoGuide
You are a reasoning-capable assistant agent that learns from past experience to solve problems more effectively.
You will refer to the guidelines that incorporates lessons from previous similar situations.

    At each time step, You need to generate one action given the current observation.
    You will be equipped with the following resources:
    1. A list of action guidelines with valuable guidelines.
    2. The intent of the task, which is the objective/goal that should be achieve.
    3. Trajectory history, which includes the current observation and a sequence of past actions.

    <instructions>
    - Not all guidelines are useful to generate the next action. Please select the guidelines that are useful and relevant
to the next action given the current observation and past actions.
    - To generate the next action, think it carefully that which guidelines from the provided guidelines are most useful
to directly tell you what to do for the next action.
    </instructions>
```

Figure 4: Prompts of all baselines on ToolSandbox.

succeeding on the first attempt. This suggests that the proposed three-pass procedure induces higher-quality workflows that are more accurate and better aligned with the prior successful experiences, allowing the agent to solve tasks without relying on iterative trial-and-error.

A.3.3 Investigation of Generalization Across Tasks

To assess whether workflows induced by WISE-Flow generalize across tasks, we evaluate cross-task reuse of retrieved workflows. As shown in Section 5.1, mixing experience from multiple tasks can be beneficial. In this experiment, agents retrieve workflows from a pool of experiences collected from other tasks, where we explicitly exclude the experience from the same task. We evaluate whether cross-task borrowing can still improve performance. This experiment aims to test the transferability and practical value of structured workflows beyond within-task reuse.

From Figure 5, we observe that WISE-Flow remains competitive even when retrieving workflows only from other tasks. Although excluding same-task experience leads to a clear drop compared with standard WISE-Flow, WISE-Flow (LOTO) still outperforms several baselines in both overall success rate and one-shot success rate, such as Reflexion

Model	Method	Similarity ↑	F_β ↑	MMR ↓	SR↑
Qwen3-235B-A22B	LLM-only	0.0647	0.0652	5.96%	12.27%
	ReAct	0.0661	0.0628	6.32%	13.38%
	Reflexion	0.1428	0.1439	13.88%	19.60%
	REMEMBERER	0.0586	0.0558	5.64%	13.25%
	AutoGuide	0.0359	0.0327	3.31%	9.70%
	AWM	0.0750	0.0754	7.71%	16.67%
	WISE-Flow (Ours)	0.0445	0.0382	3.81%	10.46%
Claude 3.7 Sonnet	LLM-only	0.0339	0.0322	3.20%	5.01%
	ReAct	0.0445	0.0413	4.15%	8.74%
	Reflexion	0.0766	0.0755	7.64%	16.12%
	REMEMBERER	0.0228	0.0212	2.09%	5.02%
	AutoGuide	0.0196	0.0131	1.32%	3.58%
	AWM	0.0408	0.0395	4.01%	8.04%
	WISE-Flow (Ours)	0.0353	0.0334	3.36%	7.68%

Table 5: Standard deviation of the results in Table 1.

Model	Method	pass^1	F_β	MMR
Qwen3-235B-A22B	LLM	0.2983	0.2343	0.2367
	ReAct	0.2189	0.2381	0.2379
	Refl.	0.2228	0.1554	0.1551
	REM	0.2696	0.2224	0.2215
	AutoG	0.1925	0.1620	0.1613
	AWM	0.2937	0.2193	0.2213
	WISE (Ours)	0.2482	0.1447	0.1445
Claude3.7 Sonnet	LLM	0.2037	0.1628	0.1632
	ReAct	0.3283	0.2541	0.2547
	Refl.	0.1773	0.1536	0.1528
	REM	0.1722	0.1541	0.1542
	AutoG	0.1519	0.1430	0.1430
	AWM	0.2127	0.1821	0.1817
	WISE (Ours)	0.1671	0.1216	0.1217

Table 6: Standard deviation of the results in Table 2. (LLM = LLM-only, Refl. = Reflexion, REM = REMEMBERER, AutoG = AutoGuide, WISE = WISE-Flow.)

	One-Pass (Single-shot)	Three-Pass
Similarity ↑	0.8659	**0.8824**
F_β ↑	0.9256	**0.9433**
MMR ↓	7.44%	**5.77%**
SR ↑	84.63%	**87.50%**
SR in one-shot ↑	65.88%	**86.25%**
SR with trial-and-error ↓	18.75%	**1.25%**

Table 7: Comparison of one-pass (single-shot) and three-pass (analyze–draft–reflect&revise) workflow induction. Experiments are conducted on ToolSandbox with Claude 3.7 Sonnet as the base model, using trajectory-wise ground-truth-based retrieval. For each test task, we retrieve all workflows induced from all trajectories of the same task, eliminating retrieval randomness and workflow aggregation noise, and isolating the effect of the induction procedure.

and AutoGuide. This suggests that the structured workflows induced by WISE-Flow capture reusable patterns that can transfer across tasks.

A.3.4 Effectiveness of WISE-Flow Retrieval

To verify that WISE-Flow benefits from retrieving relevant workflows, rather than arbitrarily selecting from the offline-induced workflow pool, we evaluate a variant of the WISE-Flow agent that randomly retrieves workflows from the same pool and compare it with the standard WISE-Flow which uses similarity-based retrieval. As shown in Table 8, the random-retrieval variant exhibits consistent degradations across all metrics, with the overall success rate decreasing by 3% and the one-shot success rate (i.e., success without any intermediate failures) dropping by 15%. These results indicate that WISE-Flow's gains are driven by retrieval quality. The agent retrieves workflows that are genuinely relevant for the query, whereas randomly selected workflows cannot achieve comparable performance.

A.3.5 Hyperparameter Analysis

In the WISE-Flow retrieval stage, the number of retrieved workflows, denoted by K, is the pri-

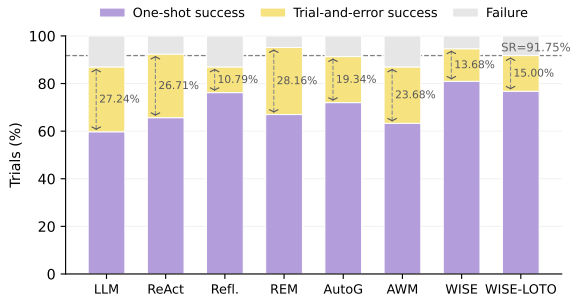


Figure 5: Evaluation of generalization across tasks on ToolSandbox. We compare the outcome composition of different methods, decomposing trials into one-shot success, trial-and-error success, and failure. WISE-Flow leave-one-task-out (WISE-LOTO) excludes the same-task experience from the retrieval pool.

Retrieval	Similarity-Based	Random
Similarity ↑	**0.9099**	0.9044
F_beta ↑	**0.9745**	0.9660
MMR ↓	**2.46%**	3.21%
SR ↑	**94.61%**	91.45%
SR in one-shot ↑	**80.92%**	65.79%

Table 8: Performance comparison of WISE-Flow using standard similarity-based retrieval versus random retrieval on ToolSandbox.

mary configurable hyperparameter, which keeps the framework simple and easy to tune. We conduct a sensitivity analysis on K by evaluating WISE-Flow on ToolSandbox with Claude 3.7, varying $K \in \{1, 2, 3, 4, 5\}$. As shown in Table 9, the performance exhibits a clear trend, with the F_β score peaking at $K = 3$. When K is too small, the retrieved experiences lack sufficient diversity to cover complex task variants. In contrast, when $K > 3$, retrieving excessive workflows introduces noise and distracts the model with less relevant context, leading to a slight performance degradation.

Based on this empirical observation, we fix $K = 3$ across all experiments to ensure a strictly fair comparison, keeping the number of retrieved experiences consistent across methods. It is also practically important because some baselines, such as those retrieving raw interaction traces, produce substantially longer textual contexts. Using a larger value of K would therefore risk exceeding the context-window limits of the evaluated models.

K	1	2	3	4	5
F_β	0.9469	0.9577	**0.9745**	0.9616	0.9584

Table 9: Hyperparameter study on the number of retrieved workflows K.

A.4 Misc

We verified that our use of third-party artifacts was consistent with their stated intended use whenever such guidance was provided. In particular, we used existing models, benchmarks, and datasets solely for their intended purpose and research purpose, and we followed any licensing and usage constraints (including platform terms for hosted models). We did not use these artifacts in ways that contradict their documented scope.

For artifacts we create, we specify the intended use as research evaluation and reproducibility. We ensure compatibility with the original access conditions of upstream resources: we do not redistribute proprietary model artifacts or restricted third-party materials, and we avoid releasing raw dataset contents or any derivatives that would violate upstream terms. If any data were accessed under research-only conditions, any derived outputs are used and shared only within research contexts, consistent with those conditions.

A.4.1 License

All proprietary models used in our experiments are used under the AWS Customer Agreement and AWS Service Terms; where applicable, Amazon Bedrock serverless third-party models are additionally subject to the relevant third-party model EULA/terms listed by AWS. Specifically, Qwen3-235B-A22B uses Apache License 2.0, Claude 3.7 Sonnet is Proprietary (Anthropic Terms of Service), and cohere.embed-english-v3 is Proprietary (Cohere / AWS Bedrock Terms of Use).

Meanwhile, the datasets and the benchmarks used are licensed as follows: (1) τ^2-bench: MIT License; (2) ToolSandbox: Apple Software License (https://github.com/apple/ToolSandbox/blob/main/LICENSE).

A.4.2 Model Usage

Qwen3-235B-A22B is a MoE model with 235B total parameters. For Claude 3.7 Sonnet and cohere.embed-english-v3, the providers do not publicly disclose parameter counts in their documentation, so we report the model identifiers.

Because inference was performed via AWS Bedrock APIs and we did not log aggregate usage (e.g., total requests/tokens/cost), we cannot report a precise total computational budget (GPU-hours). The underlying accelerator hardware for Bedrock-hosted inference is not exposed to users.

```
# Step 1 Analyze

<INSTRUCTION>
    <task_definition>
    You are given multiple user-agent dialogues. All of them represent the same user intent (task scenario), but may differ in their exact actions, wording, order, mistakes, or retries.

    Your task is to analyze and **synthesize** the most complete and representative **workflow** for this scenario, capturing all essential steps and valid branches that appear across conversations. You must include all legitimate alternative paths while excluding trial-and-error actions or error paths.
    </task_definition>

    <input_data>
    # --- Reference Conversations ---

    # {''.join(conversation_str_list)}

    # --- Reference Conversations End---
    </input_data>

    <analysis_instructions>
    ## Step 1: ANALYZE DIALOGUES

    Before generating any structured output, thoroughly analyze all dialogues:

    For each conversation:
    - Identify the user's primary goal and how the agent attempted to achieve it
    - Map out all key **action steps** taken
    - Categorize each step as:
    - **Essential**: Required steps that directly contribute to task completion
    - **Alternative path**: Valid alternative approaches to achieve the same goal
    - **Decision point**: Where the workflow branches based on conditions or choices
    - **Error path**: Steps resulting from mistakes, misunderstandings, or incorrect usage
    - **Retry**: Repeated attempts after failures

    Pay special attention to:
    - The most direct and efficient paths to goal completion
    - Valid alternative approaches that successfully complete the task
    - Common prerequisites that must be satisfied before certain actions
    - Decision points where the workflow legitimately branches
    - Steps that consistently lead to errors or failures (to be excluded)

    Synthesize across all dialogues:
    - The complete task workflow including all valid paths and branches
    - The optimal/most efficient path to completion
    - Analyse common failure points and their causes and avoid them in planned steps
    - Prerequisites for each action step
    </analysis_instructions>
</INSTRUCTION>
```

(a) Analyze

```
# Step 2 Draft
<INSTRUCTION>
    You have already analysed the conversation. Based on the above analysis, let' move on to step 2.

    <workflow_structure>
    ## Step 2: DRAFT WORKFLOW STRUCTURE
    Generate the following two parts:

    ### Part 1: Workflows

    Each **Workflow** represents a complete user task (e.g., "notify user when battery is low", "check and diagnose robot health").
    A workflow is composed of reusable action blocks.

    Each workflow should include:
    - `name`: A short, unique identifier.
    - `description`: A concise natural-language summary of the task and scenario.
    - `global_prerequisite`: A list of conditions that must be satisfied before this workflow can begin. These apply regardless of specific steps.
    - `entry_steps`: A list of action block names that may serve as entry points for this workflow.
    - `planned_steps`: A list of all action blocks involved in the workflow (including entry steps and downstream steps).

    > Use `planned_steps` to express all valid steps that belong in the task, including alternative paths.
    > Include only steps that contribute to successful task completion - exclude error paths and trial-and-error actions.

    ### Part 2: Action Blocks
    Each Action Block represents a reusable step (e.g., tool call, logic step) used in workflows.

    Each action block should include:

    - `name`: Unique identifier.
    - `is_tool_call`: Whether this action is a tool/API/function call (true or false).
    - `description`: Clear explanation of what the action does.
    - `global_prerequisite` (optional): Conditions always required for this action, regardless of workflow.
    - `scenario_prerequisite` (optional): Workflow-specific prerequisites — a mapping from workflow name to conditions specific to that usage context.
    - `next_steps` (optional): A mapping from workflow name to a list of possible next actions.
      - Each item in the list can include:
        - `name`: Action block name to transition to.
        - `condition` (optional): Logic condition to transition (e.g., "battery_level < 20").
        - `required` (optional): If the transition must occur (true or false).

    > For decision points: Use `next_steps` to represent legitimate branching paths based on different conditions
    > For alternative paths: Include all valid alternatives that achieve the same goal
    > Exclude: Steps that consistently lead to errors or require retries
    </workflow_structure>

    </workflow_draft_format>{WORKFLOW_FORMAT}</workflow_draft_format>
</ INSTRUCTION>
```

(b) Draft

```
# Step 3 Reflect & Revise

<INSTRUCTION>
    You have already drafted the workflow. Based on the above draft, let' move on to step 3.

    <reflection_guidelines>
    ## Step 3: REFLECT AND REFINE

    Critically review your draft output, carefully examining all steps and transitions:
    - What to do to solve the failures?
    - Moving decision points ahead to avoid any failures.
    - **Excludes** steps that consistently failed due to:
      - Unmet prerequisites
      - Incorrect usage
      - Misunderstanding of requirements
      - Trial-and-error attempts

    - Includes at least one complete valid execution path from entry to goal
    - Covers **all valid action paths and branches that contains no errors** across the conversations
    - Properly represents decision points where the workflow legitimately branches
    - Clearly defines prerequisites for actions that require specific conditions
    - Avoids duplications and eliminates ineffective detours
    - Makes all `planned_steps` reachable from `entry_steps` under valid conditions
    - Ensures all `next_steps` transitions respect prerequisites of the target action

    Modify/add the planned steps and prerequisites in both of the workflow blocks and action blocks.

    For each action block, verify:
    - It represents a distinct, meaningful step in the workflow
    - Its prerequisites are clearly defined and put at the propoer positions. (global_prerequisite is for scenario-agnostic prerequisite and scenario_prerequisite is for scenarios-dependent prerequisite.)
    - Its relationship to other blocks is properly represented
    - It belongs to a valid path toward goal completion

    If you find inconsistencies or errors, revise the structure accordingly.
    </reflection_guidelines>
</INSTRUCTION>


<INSTRUCTION>
    You have already reflected on the draft workflow. Based on the above reflection, let' move on to step 4.
     ## Step 4: FINALIZE

    Output your final result. Your output should strictly follow the following format:
    <output_format>
    {WORKFLOW_FORMAT}
    </output_format>

    Please return only valid JSON, without any extra text or formatting. No explanations, no comments, just pure JSON.
</INSTRUCTION>
```

(c) Reflect and revise

```
# WISE-Flow Workflow Format
    {{
      "workflows": [
        {{
        "name": "workflow_name",
        "description": "Description of the task this workflow achieves.",
        "global_prerequisite": [
          {{
          "description": "Condition to start the workflow",
          "prerequisite": ["condition_name", "expected_status"]
          }}
        ],
        "entry_steps": ["action_block_name_1"],
        "planned_steps": [
          "action_block_name_1",
          "action_block_name_2"
        ]
        }}
      ],
      "action_blocks": [
        {{
        "name": "action_name",
        "is_tool_call": true or false,
        "description": "Description of the task the action achieves.",
        "global_prerequisite": [
          {{
            "Name": "prerequisite_name",
            "Status": "expected result"
          }}
        ],
        "scenario_prerequisite": {{
          "scenarios_name": [
            {{
              "Name": "prerequisite_name",
              "Status": "expected result"
            }}
          ]
        }},
        "next_steps": {{
          "scenarios_name": [
            {{
              "name": "next_steps_name",
              "condition": "logic condition to transition",
              "required": true or false,
            }}
          ]
        }}
        }}
      ]
    }}
```

(d) Workflow Format

Figure 6: Offline Workflow Induction.

```
# WISE-Flow
You are a customer service agent that helps the user according to the <policy> below.
You are tool-using: in each turn you may EITHER
- send a natural language reply to the user, OR
- invoke exactly one tool.
You cannot do both at the same time.
You will sometimes receive an <Observation> block that includes:
- The latest user or tool message
- A <past_experience> section with retrieved workflows and action suggestions
  from previous similar tasks.

You must interpret <past_experience> as follows:

1. Past workflows are guidance, not ground truth
   - They show how similar problems were solved before.
   - They may contain workflows, planned steps, and candidate next actions.
   - You MUST still follow the current <policy> and the current conversation.

2. Explicit use of prerequisites
   - Each candidate action in <past_experience> may list:
     - satisfied_prereq: prerequisites that are already satisfied in the current conversation
     - unsatisfied_prereq: prerequisites that are NOT yet satisfied
   - Before you call a tool or decide an action, you MUST:
     - Check whether any unsatisfied prerequisites exist for that action.
     - If important prerequisites are unsatisfied, you SHOULD NOT execute that action directly.
     - Instead, you should:
       - First execute actions that help satisfy those prerequisites (such as calling other tools),
         OR
       - Ask the user for missing information, if the prerequisite requires user-provided data.

3. Choosing next actions
   - When several candidate actions are available from <past_experience>, prefer actions that:
     - Are aligned with the user's current goal AND
     - Have fewer unsatisfied prerequisites AND
     - Do not conflict with the current <policy>.
   - If no candidate action is clearly suitable, you may ignore the suggestions
     and plan the next step yourself, still respecting prerequisites and policy.

4. Safety and consistency
   - If <past_experience> suggests an action that conflicts with the <policy>,
     you MUST ignore that suggestion and follow the policy instead.
   - Do not rely on hidden IDs like task_id or internal index numbers.
     Only use human-readable workflow names, descriptions, and action names.
   - If <past_experience> explicitly states that no similar workflows were found,
     you should solve the problem normally using your general reasoning and tools.

Always:
- Follow the <policy>.
- Make tool calls only in the required JSON format.
- Be clear and concise in user-facing messages.
```

Figure 7: The system Prompt for WISE-Flow deployment on τ^2-bench.

```
# WISE-Flow:

## Prerequisites Check
You are a strict prerequisite checker.

    Overall task (what we are trying to solve):: {workflow_name}

    Actions already executed (trace, in order): {executed_actions}

    Next action we want to execute (candidate action):
    {json.dumps(action_block, ensure_ascii=False, indent=2)}

    Prerequisites that must be satisfied BEFORE executing the next action:
    Global prerequisites (evaluate all):
    {json.dumps(global_pr, ensure_ascii=False, indent=2)}

    Scenario prerequisites mapping (scenario_name -> prerequisites):
    {json.dumps(scenario_pr_all, ensure_ascii=False, indent=2)}

    Your job (two steps):
    1) Pick the SINGLE most appropriate scenario_name from the scenario prerequisites mapping,
       based on the overall task + executed actions trace + next action.
    2) Check whether the prerequisites for the NEXT action are satisfied by the executed actions trace, including:
       - all Global prerequisites
       - ONLY the prerequisites under the selected scenario_name

    Rules:
    - Do NOT assume scenario_name == workflow_name.
    - Use ONLY evidence from the executed actions trace. If the trace does not clearly show it, mark it UNSATISFIED.
    - Do NOT assume anything that is not in the trace.
    - Evaluate EVERY prerequisite item in both lists (global + scenario). Do not skip any.
    - If you cannot confidently pick a scenario_name, or there is only one scenario in Scenario prerequisites mapping,
pick it and check whether it is satisfied.

    Output (brief):
    Satisfied prerequisites:
    - <prerequisite>: <briefly describe what has been satisfied>

    Unsatisfied prerequisites:
    - <prerequisite>: <briefly describe what is missing / what needs to happen>

## Summarize the task
Your job is to analyze user utterances and extract the key information to create clear, concise, organized task summaries
that capture the main request, important details, and relevant context.

<user_message>
{joined}
</user_message>

Please write a concise task summary and enclose it in <output></output> tags.
```

Figure 8: During WISE-Flow online deployment, we use an LLM (the same base model as the agent) to summarize the task from the current dialogue and to check whether the prerequisites of the retrieved actions are satisfied. The figure shows the two prompts used for task summarization and prerequisite checking.

```
"task_id": "search_message_with_recency_latest_alt",
"workflows": [
    [{
       "name": "retrieve_latest_text_message",
       "description": "Retrieve and read the latest text message from the user's message history.",
       "global_prerequisite": [
        {
         "description": "User must have at least one text message in their history",
         "prerequisite": [ "message_history", "exists"]
        },
        {
         "description": "System must have permission to access text messages",
         "prerequisite": ["message_access_permission", "granted"]
        }
       ],
       "entry_steps": ["get_latest_message_timestamp"],
       "planned_steps": [
        "get_latest_message_timestamp",
        "retrieve_message_content",
        "present_message_to_user"
       ]
      }
    ]
],
"action_blocks": [
    [ ...
     {
       "name": "retrieve_message_content",
       "is_tool_call": true,
       "description": "Retrieve the full message content using the timestamp of the latest message.",
       "global_prerequisite": [
                {"Name": "message_history",  "Status": "exists"},
                {"Name": "message_access_permission",  "Status": "granted"}
       ],
       "scenario_prerequisite": {
                 "retrieve_latest_text": [{"Name": "latest_message_timestamp", "Status": "identified"}]
       },
       "next_steps": {
                "retrieve_latest_text": [
                        {
                          "name": "present_message_to_user",
                          "condition": "message_content_retrieved",
                          "required": true
                        }
                ]
       }
      },
      ...
    ]
]
```

Figure 9: A workflow sample generated by WISE-Flow during offline workflow induction on ToolSandbox using Claude 3.7 Sonnet.