Intent-Driven Situation Tracking for User-Centric Multi-Turn Agents

Meiling Tao[1]*, Yiling Tao[2]*, Peng Wang[1]†
[1]University of Electronic Science and Technology of China
[2]Shenzhen International Graduate School, Tsinghua University
meilingtao@std.uestc.edu.cn, taoyl25@mails.tsinghua.edu.cn, p.wang6@hotmail.com

Abstract

User-centric multi-turn agents must act on an evolving task situation shaped by changing user intents, accumulated tool-grounded facts, missing information, and execution constraints. Existing context-management methods improve the use of past interaction history, but rarely maintain an explicit situation state that separates grounded facts from task-state judgments. As a result, agents often need to infer fine-grained attributes, task dependencies, and constraint satisfaction implicitly from dialogue traces. We propose Intent-Driven Situation States (IDSS), a training-free framework that maintains an explicit situation state alongside the dialogue. IDSS parses tool returns into provenance-aware entities and attributes, tracks user intents, required variables, constraints, and execution status, and propagates new facts to task constraints to update action executability. This allows agents to avoid infeasible actions, advance dependent goals, and reuse relevant information without repeatedly searching raw history. Experiments on three interactive benchmarks across eight LLMs show that IDSS improves task completion, preference elicitation, and interaction efficiency, with clear gains on tasks involving multi-entity coordination, evolving user constraints, and constraint-aware replanning. Ablations and error analyses show that these improvements come from the interaction between fact persistence, intent-centered state tracking, and constraint modeling. These results suggest that explicit situation tracking offers an effective alternative to history-centric context management for reliable user-centric multi-turn agents.

1 Introduction

LLM-based agents are increasingly used in user-facing applications such as customer service, travel planning, and life services (Schick et al., 2023; Qin et al., 2024; Patil et al., 2024). In these scenarios, agents no longer execute isolated instructions (Wei et al., 2022; Yao et al., 2023), but instead engage in sustained interactions where they must understand evolving user goals, call external tools, follow rules, and decide when to ask, retrieve, execute, or finish. User-centric multi-turn tasks therefore place stronger demands on state maintenance than single-shot tool use or single-goal long-horizon planning.

The difficulty comes from the evolving nature of user interactions. Users may introduce multiple goals in one conversation, and these goals can depend on or conflict with one another. Preferences are often released gradually rather than stated upfront. Tool returns may revise previously assumed facts, and newly observed facts may make an earlier action path infeasible. Completing such tasks requires agents to track the current task situation: which facts are grounded, which intents are active or completed, which variables are still missing, and which constraints are satisfied or violated. Without such tracking, an agent must repeatedly reconstruct the current situation from a growing dialogue history, which can lead to fact omission, intent drift, premature execution, and constraint violations. Recent user-interaction benchmarks (Yao et al., 2024; He et al., 2025; Qian et al., 2025a) show that even strong models struggle with these issues in realistic multi-turn settings.

Prior work addresses long interactions mainly through external memory or context compression (Packer et al., 2023; Chhikara et al., 2025; Rasmussen et al., 2025; Wu et al., 2025; Ye et al., 2025; Su et al., 2026). External memory systems make interaction information retrievable beyond the immediate context window, while compression, summarization, and folding methods shorten or restructure previous turns to fit the prompt. These approaches reduce information access and context-length pressure, but the agent must still infer the current task situation from retrieved memories, summaries, or

*Equal contribution.
† Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 30432–30449
October 24-29, 2026 ©2026 Association for Computational Linguistics

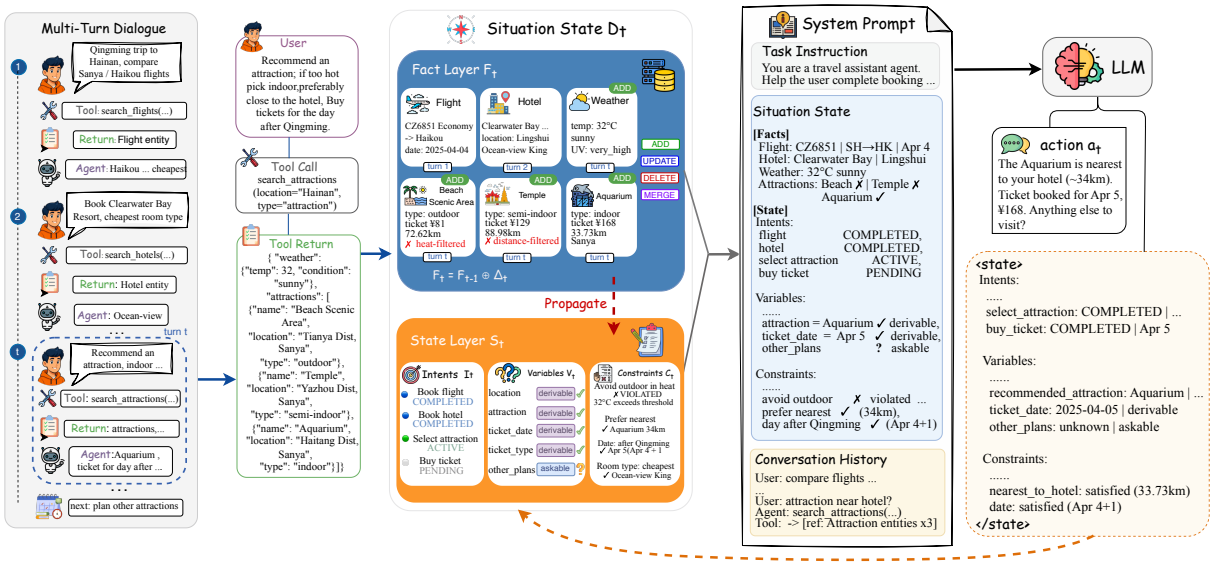


Figure 1: Overview of the IDSS framework. At each turn, the agent receives a user message or tool return: the fact layer converts tool returns into structured entities via deterministic parsing, while the state layer is updated by the agent during reasoning to reflect intents, variables, and constraints. New facts trigger cross-layer constraint propagation (red arrows), which may block infeasible intents and activate subsequent ones. Left: multi-turn dialogue timeline. Right: prompt assembly and LLM output. Blue arrows indicate tool returns parsed into the fact layer. Orange arrows indicate the updated state block being fed back into the next prompt.

compressed traces. Grounded facts are rarely organized separately from task-state judgments such as active intents, missing variables, and constraints, leaving fine-grained attributes, task dependencies, and constraint satisfaction to implicit model inference.

To address this gap, we propose **Intent-Driven Situation States** (IDSS), a training-free framework for situation tracking in user-centric multi-turn agents. IDSS maintains an explicit *situation state* alongside the dialogue history. The fact layer parses tool returns into provenance-aware entities and attributes, while the state layer tracks user intents, required variables, constraints, and execution status. Cross-layer constraint propagation links newly observed facts to task constraints, allowing the agent to block infeasible intents, activate subsequent goals, and reuse relevant information without repeatedly searching raw history. At each turn, the situation state is rendered into the prompt together with a compact residual dialogue history, providing the agent with a decision-oriented view of the current task situation, as shown in Figure 1.

We evaluate IDSS on τ-bench, VitaBench, and UserBench across eight LLMs. Our contributions are as follows:

- We formulate user-centric multi-turn interaction as intent-driven situation tracking and propose IDSS, which maintains explicit situation states separating grounded facts from task-state judgments.
- We design a dual-layer update mechanism requiring no additional LLM calls: the fact layer updates incrementally through deterministic parsing, and the state layer is maintained during action generation to track intents, variables, constraints, and execution status.
- Experiments on three interactive benchmarks across eight LLMs show that IDSS improves task completion, preference elicitation, and interaction efficiency, with the clearest gains on tasks requiring multi-entity coordination and evolving constraints.

2 Related Work

2.1 User-Centric Multi-Turn Agents

Recent benchmarks show that single-turn performance does not reliably predict multi-turn capabilities (Xi et al., 2025; Wang et al., 2024). τ-bench and τ^2-bench (Yao et al., 2024; Barres et al., 2025) test policy-following and tool-use consistency in customer-service settings, VitaBench (He et al., 2025) stresses temporal and spatial reasoning and shifting intents across life-service domains, and

UserBench (Qian et al., 2025a) focuses on incremental preference elicitation. Even strong models still struggle on these benchmarks, revealing systematic challenges in information persistence, intent tracking, and holistic task planning as conversations grow longer. On the method side, Sun et al. (2025) train proactive interaction agents via instruction tuning, UserRL (Qian et al., 2025b) applies reinforcement learning with multi-turn credit assignment, and Suri et al. (2025) model interactions as a POMDP for structured clarification. While effective, these approaches often rely on dedicated supervision, training procedures, or task-specific assumptions, which may limit cross-domain generalization. IDSS requires no additional training and instead improves multi-turn interaction by restructuring context into an explicit situation state.

2.2 Agent Context Management

Existing agent context-management methods often address long-interaction challenges through external memory or in-context compression (Zhang et al., 2025). External-memory systems, such as MemGPT, Mem0, Graphiti, and Generative Agents (Packer et al., 2023; Chhikara et al., 2025; Rasmussen et al., 2025; Park et al., 2023), improve retention through working-memory management, long-term fact consolidation, temporal knowledge graphs, or reflection-based retrieval, but do not necessarily maintain an explicit current task situation. In-context compression methods such as ReSum (Wu et al., 2025) and IterResearch (Chen et al., 2025) condense interaction histories into compact reasoning states, while LLMLingua (Jiang et al., 2023a, 2024) and AutoCompressors (Chevalier et al., 2023) prune token-level redundancy. AgentFold (Ye et al., 2025) and U-Fold (Su et al., 2026) further perform more semantic compression through sub-task folding or intent-aware extraction. However, they rarely separate tool-grounded facts from task-state judgments such as intents, missing variables, and constraints, which IDSS maintains in an explicit situation state for decision-making.

2.3 State Tracking

Dialogue state tracking (DST) maintains slot-value pairs within predefined ontologies (Budzianowski et al., 2018; Heck et al., 2020), with recent extensions using LLM function calling (Li et al., 2024) or knowledge-graph reasoning (Pan et al., 2024; Jiang et al., 2023b). However, DST is not designed for the open-ended entities, dynamic constraints, and multi-intent dependencies in tool-augmented agent settings. In the agent domain, StateAct (Rozanov and Rei, 2025) proposes chain-of-states to track environment state at every step, but does not explicitly separate tool-grounded facts from user task goals, nor does it model constraint dependencies. PABU (Jiang et al., 2026) maintains a belief state via progress prediction and selective history retention, yet its linear progress assumption may be insufficient for non-linear multi-intent dependencies, and it requires fine-tuning on curated trajectories. IDSS instead separates tool-grounded facts from task-state judgments and links them through constraint propagation, allowing non-linear intent dependencies and blocking conditions to be updated across turns.

3 Method

3.1 Problem Formulation

Consider an LLM agent that assists a user through multi-turn interactions. At each turn t, the agent receives an input x_t, which can be either a user message or a tool return, and generates an output a_t, such as a tool call or a user-facing response. Let H_t denote the interaction context available when generating a_t:

$$H_t = (x_1, a_1, x_2, a_2, \ldots, a_{t-1}, x_t). \quad (1)$$

Given H_t and the available tool set $\mathcal{A}$, the agent follows a policy π:

$$a_t = \pi(H_t, \mathcal{A}). \quad (2)$$

Under the standard ReAct paradigm, H_t is maintained as a linear sequence of user messages, tool returns, and agent outputs. As interactions grow, this linear history becomes inadequate for user-centric multi-turn tasks: relevant information may be truncated or compressed, tool-grounded attributes may be buried in long traces, and preferences, tool preconditions, and rules may be scattered across turns. The agent must repeatedly reconstruct the current task situation from raw history, which can lead to fact omission, intent drift, premature execution, and constraint violations.

3.2 Intent-Driven Situation Tracking

To address these limitations, we propose **Intent-Driven Situation States (IDSS)**, a training-free framework that maintains an explicit situation state alongside the dialogue history. Rather than relying

on the agent to infer the current situation from a long, mixed sequence of user messages, tool calls, and tool returns, IDSS organizes multi-turn interaction around evolving user intents, while grounding decisions in structured facts, required variables, and execution constraints.

Formally, IDSS replaces the history-only policy with a situation-augmented policy:

$$a_t = \pi(D_t, H_t^{\text{comp}}, \mathcal{A}), \tag{3}$$

where D_t denotes the structured situation state maintained by IDSS, and H_t^{comp} is the compact residual history obtained from H_t after state-covered information is removed or replaced with references to D_t. In practice, H_t^{comp} retains recent user-agent turns while replacing parsed tool outputs with references to entries in the fact layer. Through this separation, the agent acts on an explicit representation of the current task situation together with a lightweight dialogue trace, rather than searching through unorganized raw history.

As shown in Figure 1, IDSS consists of three components: a fact layer for tool-grounded information, a state layer for user-task progress, and a cross-layer propagation step that aligns task progress with newly observed facts. Formally, the situation state is defined as:

$$D_t = (F_t, S_t), \tag{4}$$

where F_t stores structured entities and attributes extracted from tool returns, and S_t tracks user intents, required variables, constraints, and execution status.

3.2.1 Fact Layer

Tool returns across turns often describe interrelated entities, such as flights, hotels, and orders, together with attributes such as price, location, date, and status. The fact layer organizes these observations into a structured entity store, enabling cross-entity and cross-turn querying without tracing back through raw conversation history.

Extraction and Update. The fact layer is defined as:

$$F_t = \{e_i = (\text{type}_i, \text{id}_i, \text{attrs}_i, \text{src}_i)\}, \tag{5}$$

where type_i denotes the entity type, id_i the entity identifier, attrs_i the attribute set, and src_i the provenance source.

Let o_t denote a tool-return observation when x_t is produced by a tool. A deterministic parser converts o_t into an operation sequence:

$$\Delta_t = \text{Parse}(o_t), \quad \Delta_t = [\delta_1, \delta_2, \ldots], \tag{6}$$

where each δ_i is an atomic operation. The fact layer is then updated incrementally:

$$F_t = F_{t-1} \oplus \Delta_t. \tag{7}$$

Operations include `ADD`, `UPDATE` for attribute-level merging, `DELETE` for marking invalid entities, and `MERGE` for combining duplicates.

Conflict Resolution and Compression. When the same attribute receives conflicting values, the fact layer resolves them by source credibility. Tool-observed attributes are prioritized over agent-inferred ones, while user preferences and constraints are maintained in the state layer. Within the same source type, newer observations update older values. This prevents hallucinated or stale information from polluting the fact store.

After entity extraction, corresponding tool returns in the conversation history are replaced with brief fact-layer references. This keeps the prompt compact while preserving factual information in an explicit and accessible form.

3.2.2 State Layer

While the fact layer stores tool-grounded observations, the state layer tracks task-state judgments that evolve across turns. It is organized around user intents and records intent progress, dependencies, required variables, and constraints. The agent is instructed to prepend a `<state>` block to each response, listing intents, confirmed conclusions, and missing information. This block is parsed into S_t, removed before user delivery, and re-injected into the next prompt:

$$S_t = (I_t, V_t, C_t), \tag{8}$$

where I_t is the intent set, V_t the variable set, and C_t the constraint set.

The state layer also guides normal completion decisions: unless the user explicitly stops or an external turn limit is reached, the agent treats the current task as complete when all user-introduced intents are `COMPLETED` or `BLOCKED` and no required variables remain unknown for executable intents. Otherwise, remaining `PENDING` intents or unresolved variables guide the next question, retrieval, or tool action.

Intent Tracking. The agent identifies explicit or implicit task goals from user messages and records them as intents. New user requests are appended to the intent set. Each intent is assigned one of four statuses: `active`, `pending`, `completed`, or `blocked`.

Intent dependencies are inferred from task semantics and execution preconditions. If an intent requires another intent, user confirmation, tool-returned information, or satisfied constraints before execution, it is linked to the corresponding prerequisite. Once dependencies are completed, required variables are known, and relevant constraints are not violated, the intent becomes `active`; after execution, it becomes `completed`. Intent status therefore guides the next action by identifying executable goals, information gaps, and infeasible paths.

Variable Tracking. The variable set describes information slots required by current or pending intents. Each variable is annotated as `askable` if it should be obtained from the user, `retrievable` if it requires tool invocation, or `derivable` if it can be inferred from existing facts. Variables are also marked as known or unknown, helping the agent decide whether to ask, retrieve, or derive information.

Constraint Modeling. The constraint set records conditions that actions must satisfy, including user preferences, tool preconditions, and rules. Each constraint is represented as a short rule description with grounded arguments when available, such as an entity attribute, operator, and required value, and its status is marked as `satisfied`, `unsatisfied`, or `violated`. For example, in the travel-planning case in Figure 1, an attraction-selection intent may include constraints such as `nearest_to_hotel` or `avoid_outdoor`, whose status is updated when hotel-location or weather facts become available. When a constraint becomes `violated`, the associated intent is marked as `blocked`, preventing repeated attempts at infeasible actions.

3.3 Cross-Layer Constraint Propagation

The fact and state layers are connected through cross-layer constraint propagation. Whenever the fact layer updates entity attributes, the system re-evaluates grounded constraints in C_t through deterministic attribute comparison. Constraints without sufficient grounding remain `unsatisfied` rather than being incorrectly marked as `violated`. When a constraint is violated, the system blocks the associated intent and records the factual basis and constraint chain behind the decision.

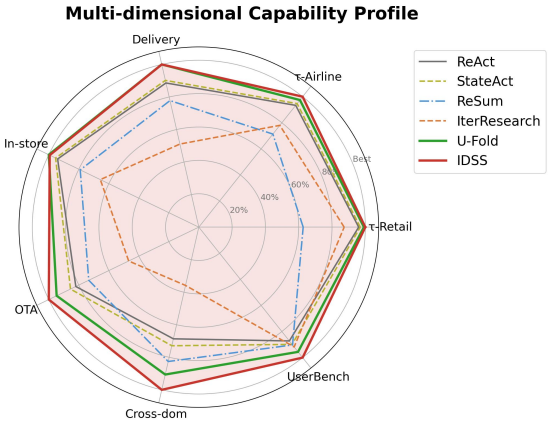


Figure 2: Normalized performance radar across seven evaluation dimensions. IDSS (red) achieves the largest coverage area.

This mechanism allows the agent to understand why a path is infeasible and derive alternatives instead of retrying failed operations. For example, a newly observed fare-class attribute may violate a flight-modification constraint, block the modification intent, and activate a cancellation-and-rebooking intent while preserving the user's original requirements. Figure 4 illustrates this mechanism on a concrete τ-bench Airline task.

4 Experiments

4.1 Experimental Setup

We evaluate IDSS on three interactive benchmarks. **τ-bench** (Yao et al., 2024) includes Retail tasks on product inquiry, order management, and returns, and a more constraint-heavy Airline domain involving booking changes, cancellation, baggage, and fare-class restrictions. We report *Avg@4* and *pass*^{4} over four trials. **VitaBench** (He et al., 2025) covers Delivery, In-store, OTA, and Cross-domain life-service tasks, where the latter two require more cross-service coordination and state inheritance, and reports rubric-based scores from 0 to 100. **UserBench** (Qian et al., 2025a) evaluates under-specified requests requiring preference elicitation, with *Score*, *CER* (Correct Exist Rate), and *PE* (Preference Elicited) as metrics.

We compare against ReAct (Yao et al., 2022), StateAct (Rozanov and Rei, 2025), ReSum (Wu et al., 2025), IterResearch (Chen et al., 2025), and U-Fold (Su et al., 2026) (see Appendix A). We

Table 1: Main results across three benchmarks with eight models. **Bold** indicates best, underline indicates second best within each model group. Row background reflects improvement over the ReAct baseline: deeper blue indicates larger gains, deeper red indicates larger degradation.

Model	Method	τ-bench				VitaBench (Avg@4)				UserBench		
		Retail		Airline								
		Avg@4	pass^4	Avg@4	pass^4	Delivery	In-store	OTA	Cross	Score	CER	PE (%)
GPT-5.5	ReAct	77.5	46.0	63.5	28.5	52.0	61.5	34.0	22.5	0.368	0.405	28.5
	StateAct	78.2	47.0	64.2	29.5	52.5	62.0	35.5	23.5	0.378	0.414	29.0
	ReSum	52.0	28.5	48.5	22.0	44.5	52.0	30.2	25.5	0.380	0.418	29.8
	IterResearch	70.5	38.5	52.5	23.5	30.0	44.5	20.0	12.0	0.385	0.422	30.2
	U-Fold	79.0	48.5	65.0	30.5	**57.0**	**64.0**	38.0	27.0	0.395	0.435	31.5
	IDSS	**80.5**	**51.0**	**66.5**	**33.0**	56.5	63.5	**40.5**	**29.8**	**0.415**	**0.455**	**33.5**
GPT-4.1	ReAct	71.8	38.0	56.0	22.0	46.0	56.2	28.5	17.8	0.330	0.365	24.2
	StateAct	72.6	39.2	56.8	23.2	46.3	56.4	30.0	18.2	0.340	0.374	24.8
	ReSum	46.5	24.0	42.5	18.0	39.5	47.0	26.2	20.8	0.342	0.378	25.5
	IterResearch	64.8	32.5	46.5	19.5	25.5	39.0	16.5	9.2	0.348	0.384	25.8
	U-Fold	73.5	40.5	57.5	24.5	**51.5**	**58.5**	32.5	22.0	0.360	0.398	27.5
	IDSS	**75.0**	**43.0**	**59.0**	**26.5**	51.2	58.0	**34.8**	**24.8**	**0.382**	**0.418**	**29.5**
GPT-4o	ReAct	68.5	34.0	52.0	18.5	42.0	52.3	26.0	14.5	0.312	0.345	22.5
	StateAct	69.0	35.2	53.0	19.5	43.0	53.0	27.2	15.5	0.322	0.355	23.2
	ReSum	44.5	21.5	40.5	15.8	37.5	43.5	23.2	17.8	0.324	0.358	23.8
	IterResearch	62.3	29.5	44.5	16.5	24.5	36.5	15.0	7.5	0.328	0.362	24.0
	U-Fold	**70.2**	37.5	54.5	21.2	48.0	**55.8**	30.5	19.8	0.345	0.380	26.0
	IDSS	70.0	**39.8**	**56.0**	**23.5**	**49.8**	55.5	**32.2**	**22.0**	**0.365**	**0.398**	**28.2**
Claude-4.5-Sonnet	ReAct	67.0	32.5	51.5	17.8	43.5	53.0	27.5	15.0	0.318	0.350	24.0
	StateAct	67.8	33.8	52.5	18.8	44.5	54.2	28.5	16.2	0.328	0.360	24.5
	ReSum	43.5	20.8	39.5	14.8	38.8	44.5	24.8	18.3	0.329	0.362	25.0
	IterResearch	61.0	28.0	43.0	15.8	25.0	37.0	15.5	7.8	0.334	0.368	25.5
	U-Fold	69.5	36.8	54.0	20.8	**49.5**	**56.5**	31.8	20.5	0.350	0.385	27.0
	IDSS	**71.0**	**38.8**	**55.5**	**23.0**	49.2	56.2	**33.5**	**22.8**	**0.370**	**0.405**	**29.0**
Gemini-2.5-Pro	ReAct	69.0	34.5	53.0	19.0	44.0	54.5	27.8	15.8	0.320	0.352	23.0
	StateAct	69.8	35.8	53.8	20.0	45.0	55.2	29.0	16.8	0.330	0.362	23.5
	ReSum	45.0	21.8	40.5	15.5	38.8	45.8	24.8	19.5	0.331	0.364	24.0
	IterResearch	62.8	29.8	44.5	17.0	25.5	37.5	15.5	8.2	0.338	0.370	24.5
	U-Fold	**71.0**	38.0	55.2	21.8	49.5	**57.5**	31.5	20.2	0.348	0.382	26.2
	IDSS	70.5	**40.2**	**57.0**	**24.0**	**50.0**	57.2	**33.5**	**23.0**	**0.370**	**0.404**	**28.5**
DeepSeek-V3.2	ReAct	66.5	32.0	51.0	17.5	42.5	52.0	26.5	14.2	0.310	0.342	22.2
	StateAct	67.2	33.0	51.8	18.5	43.5	53.0	27.5	15.5	0.320	0.352	23.0
	ReSum	43.2	20.2	39.0	14.8	37.5	43.8	23.5	17.5	0.321	0.354	23.2
	IterResearch	60.5	27.5	42.8	15.5	24.5	36.0	15.0	7.5	0.327	0.359	23.8
	U-Fold	69.0	36.0	53.5	20.5	**48.2**	**55.8**	30.2	19.5	0.340	0.374	25.2
	IDSS	**69.5**	**38.0**	**55.0**	**22.5**	47.8	55.5	**32.0**	**21.5**	**0.358**	**0.392**	**27.8**
GPT-4.1-mini	ReAct	61.5	27.0	45.5	14.2	36.8	47.5	21.5	10.8	0.285	0.315	19.5
	StateAct	62.0	28.0	46.0	14.8	37.5	48.3	22.4	11.5	0.292	0.322	20.2
	ReSum	40.0	17.0	34.5	11.8	32.5	40.0	19.0	13.3	0.295	0.326	20.8
	IterResearch	56.0	23.2	37.8	12.8	21.5	32.8	12.0	5.6	0.301	0.331	21.0
	U-Fold	64.5	**31.2**	48.5	17.2	**42.5**	**51.5**	26.0	15.8	0.318	0.350	**23.8**
	IDSS	**64.8**	30.5	**49.5**	**18.5**	42.0	51.2	**27.0**	**16.5**	**0.322**	**0.354**	23.2
Gemini-2.0-Flash	ReAct	63.0	28.5	47.5	15.2	38.5	49.0	23.0	12.0	0.295	0.326	20.5
	StateAct	63.8	29.2	48.0	16.0	39.2	50.0	24.0	13.0	0.304	0.336	21.2
	ReSum	41.0	18.0	36.0	12.8	34.0	41.2	20.5	14.8	0.305	0.337	21.5
	IterResearch	57.5	24.5	39.5	13.5	22.5	33.8	13.0	6.2	0.311	0.342	22.0
	U-Fold	66.0	32.5	50.2	18.2	**43.5**	**53.0**	27.8	17.0	0.326	0.358	**24.5**
	IDSS	**66.2**	**34.0**	**51.5**	**20.0**	43.2	52.8	**28.5**	**18.2**	**0.328**	**0.360**	24.2

use GPT-4.1 as the user simulator by following a common practice and evaluate on eight LLMs. For τ-bench and VitaBench, each task is run for four independent trials and averaged.

4.2 Main Results

Table 1 reports the main results across three benchmarks and eight LLMs. Figure 2 summarizes the normalized performance profile across evaluation axes. Overall, IDSS delivers the strongest overall performance among the compared methods and consistently improves over ReAct across tool-use, life-service, and user-centric interaction settings. The radar plot further shows that IDSS has the most balanced coverage across evaluation axes, rather than improving only a single benchmark or metric. These results suggest that explicit situation states provide a general context-management benefit by helping agents preserve tool-grounded facts, track evolving user goals, and make constraint-aware decisions across different backbone models.

On τ-bench, Airline is more challenging, as rebooking, cancellation, baggage handling, and fare-class restrictions introduce cross-turn dependencies and blocking conditions. Across the eight LLMs, IDSS improves over the strongest baseline, U-Fold, by 1.5 Avg@4 and 2.0 pass^4 points on Airline. Retail gains over U-Fold are smaller but positive,

Table 2: Ablation results of IDSS on GPT-4.1 across three benchmarks.

Configuration	τ-bench Avg@4	τ-bench pass^4	VitaBench Avg@4	UserBench Score
Full IDSS	**67.0**	**34.8**	**42.2**	**0.382**
w/o Fact Layer	64.8	31.3	39.7	0.372
w/o State Layer	65.5	32.3	39.4	0.350
w/o Constraint Modeling	66.0	33.6	40.7	0.368
Vanilla	63.9	30.0	37.1	0.330

at 0.6 Avg@4 and 1.8 pass^4 points. In contrast, ReSum and IterResearch often underperform ReAct, indicating that summary-based or retrieval-heavy context management may lose precise identifiers, statuses, and time-sensitive fields required by customer-service tasks.

On VitaBench, U-Fold is the strongest baseline, especially on the simpler Delivery and In-store subtasks, where folding completed context already preserves much of the needed information. IDSS remains competitive on these simpler settings, while showing clearer advantages on OTA and Cross-domain, with gains over U-Fold of 1.7 and 2.1 points across the eight LLMs. These two scenarios require multi-entity coordination, cross-service composition, and state inheritance, where separating grounded facts from active intents and constraints provides more reliable guidance for subsequent actions.

On UserBench, IDSS achieves the best Score and CER across all evaluated LLMs, improving over U-Fold by approximately 4.6% on Score and 4.0% on CER in relative terms. Preference elicitation gains are smaller and vary across backbones, but the stronger completion-oriented metrics suggest that IDSS better distinguishes confirmed information, missing variables, and user preferences. This helps the agent ask clarification questions that are more directly tied to task progress, rather than eliciting preferences in isolation.

4.3 Ablation Study

To understand where IDSS gains come from, we conduct ablation experiments on all three benchmarks by removing the fact layer, the state layer, or constraint modeling. Specifically, *w/o Fact Layer* removes tool-fact extraction and fact-layer rendering; *w/o State Layer* removes intent tracking, variable tracking, and constraint modeling; and *w/o Constraint Modeling* retains intent and variable tracking but removes action-precondition and blocking-reason modeling.

The results reveal complementary contributions

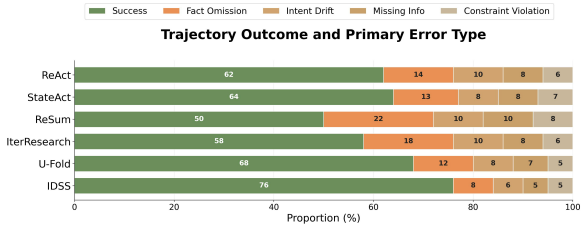


Figure 3: Trajectory outcomes and primary error types for IDSS and baselines.

from the fact and state layers. Removing the fact layer hurts τ-bench most, reducing Avg@4 by 2.2 points and pass^4 by 3.5 points, because these tasks depend heavily on persistent tool-grounded facts such as orders, fares, and reservations. Removing the state layer has the largest impact on UserBench, dropping Score by 8.4%, versus 2.6% without the fact layer, reflecting the importance of intent progress and missing-variable tracking for preference elicitation. On VitaBench, the two layers contribute similarly, with drops of 2.5 and 2.8 points, indicating that complex life-service tasks require both factual grounding and evolving user-goal tracking. Constraint modeling provides smaller but consistent gains by translating factual changes into executable or blocked intents.

4.4 Error Analysis

To better understand the error patterns behind the aggregate results, we annotate τ-bench and VitaBench trajectories into the outcome and error categories shown in Figure 3. IDSS substantially reduces errors related to state maintenance. In particular, fact omission decreases from 14% to 8%, reflecting the benefit of persistent entity storage in the fact layer. Missing-information errors also decrease from 8% to 5%, consistent with the state layer's explicit tracking of missing variables.

Different baselines exhibit distinct failure patterns. ReSum has the highest fact-omission rate at 22%, suggesting that summarization can discard fine-grained fields needed for later decisions. StateAct reduces intent drift, but its fact-omission rate remains close to ReAct because it does not explicitly preserve tool-grounded attributes. In contrast, constraint violation changes only modestly across methods, ranging from 5% to 8%. This suggests that IDSS is most effective at mitigating state-maintenance errors, while some rule violations still arise from the model's imperfect rule understanding rather than from missing state alone.

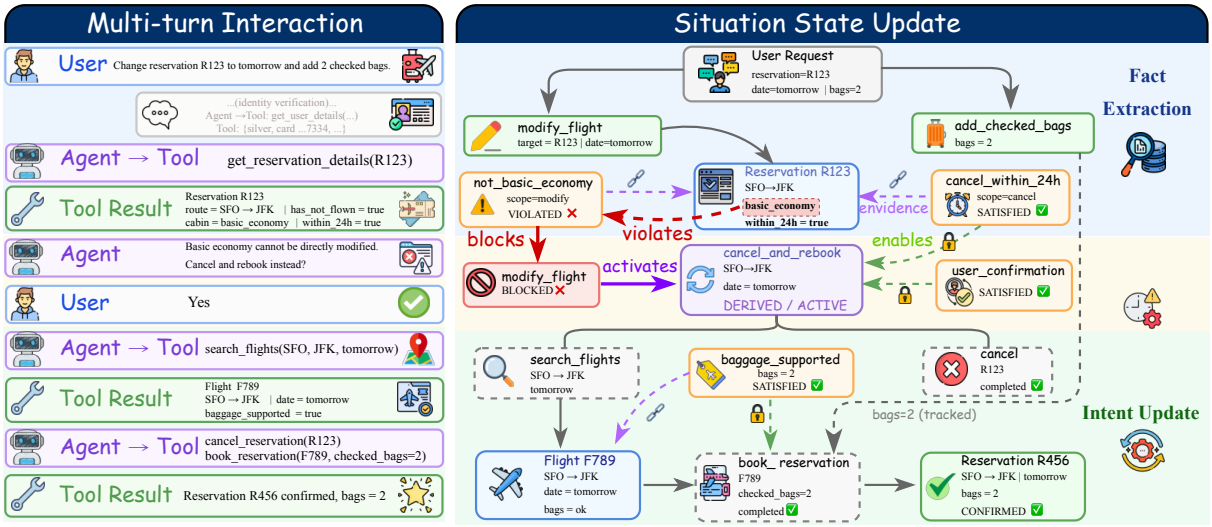
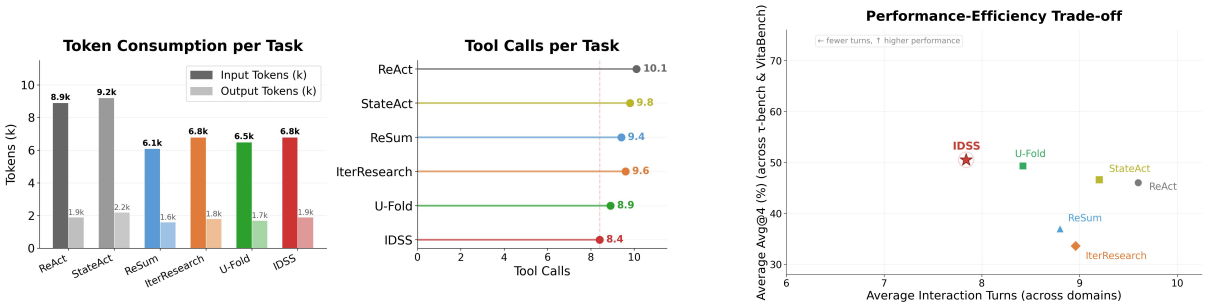


Figure 4: Case study on a τ-bench Airline task. IDSS propagates a tool-grounded fare-class fact to block an infeasible modification path and activate a feasible alternative while preserving the user's baggage requirement.



(a) Average input/output token usage and tool calls per task.

(b) Performance versus interaction turns.

Figure 5: Efficiency comparison of IDSS and baselines.

4.5 Case Study

Figure 4 presents a τ-bench Airline case where a `basic_economy` fare-class fact must be linked to the modification constraint. History-only or summary-based agents may preserve the user's request but miss the blocking relation between `basic_economy` and `modify_flight`. IDSS records this fact during extraction, blocks the infeasible `modify_flight` intent through constraint propagation, and activates an alternative path while preserving the checked bag requirement.

4.6 Efficiency Analysis

Figure 5a compares token consumption and tool-call counts, while Figure 5b plots average performance against interaction turns. IDSS keeps the prompt compact by rendering the current situation state with compressed history, requires *zero* extra LLM calls, and achieves the highest average performance (50.5%) with the fewest turns (7.8). These gains come from preserving reusable facts and preventing actions on blocked or underspecified intents.

5 Conclusion

In this work, we presented IDSS, a training-free framework for intent-driven situation tracking in user-centric multi-turn agents. IDSS maintains the current task situation as an explicit state, separating tool-grounded facts from evolving task-state judgments such as intents, missing variables, constraints, and execution status. Across three interactive benchmarks and eight LLMs, IDSS improves task completion, preference elicitation, and interaction efficiency, especially on tasks with multi-entity coordination and evolving constraints. These results suggest that reliable user-centric agents benefit from maintaining an explicit, decision-oriented situation state rather than relying only on access to past interaction history.

Limitations

IDSS is designed for interactive tasks where tool returns expose textual fields that can be converted into structured facts. Extending the same situation-state design to more diverse output formats, richer tool environments, or multimodal observations is an interesting direction for future work. Our experiments use simulated users for reproducibility and controlled comparison across models. Future studies with real users could further validate how IDSS supports naturally expressed goals, corrections, and preferences.

Ethical Considerations

This work studies training-free context management for user-centric multi-turn agents using public or simulated benchmark environments. We do not collect personal data or conduct experiments with human participants. IDSS may improve the reliability of tool-using agents, but it does not eliminate risks from incorrect tool outputs, model hallucinations, or inappropriate deployment in high-stakes settings. Practical deployments should include task-specific safeguards, logging, and human oversight when agent actions may affect users.

Acknowledgments

This work was supported in part by the Sichuan Science and Technology Program under Grant 2025YFNZH0034.

References

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*.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gasic. 2018. Multiwoz-a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 5016–5026.

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 with interaction scaling. *arXiv preprint arXiv:2511.07327*.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3829–3846.

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*.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, and 1 others. 2025. Vitabench: Benchmarking llm agents with versatile interactive tasks in real-world applications. *arXiv preprint arXiv:2509.26490*.

Michael Heck, Carel van Niekerk, Nurul Lubis, Christian Geishauser, Hsien-Chin Lin, Marco Moresi, and Milica Gasic. 2020. Trippy: A triple copy strategy for value independent neural dialog state tracking. In *Proceedings of the 21th annual meeting of the special interest group on discourse and dialogue*, pages 35–44.

Haitao Jiang, Lin Ge, Hengrui Cai, and Rui Song. 2026. Pabu: Progress-aware belief update for efficient llm agents. *arXiv preprint arXiv:2602.09138*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023a. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 13358–13376.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023b. Structgpt: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 9237–9251.

Zekun Li, Zhiyu Chen, Mike Ross, Patrick Huber, Seungwhan Moon, Zhaojiang Lin, Xin Luna Dong, Adithya Sagar, Xifeng Yan, and Paul A Crook. 2024. Large language models as zero-shot dialogue state tracker through function calling. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8688–8704.

OpenAI. 2025. Introducing gpt-4.1 in the api. https://openai.com/index/gpt-4-1/. Accessed: 2026-05-26.

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

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599.

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.

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*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*.

Nikolai Rozanov and Marek Rei. 2025. Stateact: Enhancing llm base agents via self-prompting and state-tracking. In *Proceedings of the 1st Workshop for Research on Agent Language Models (REALM 2025)*, pages 367–385.

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.

Jin Su, Runnan Fang, Yeqiu Li, Xiaobin Wang, Shihao Cai, Pengjun Xie, Ningyu Zhang, and Fajie Yuan. 2026. U-fold: Dynamic intent-aware context folding for user-centric agents. *arXiv preprint arXiv:2601.18285*.

Weiwei Sun, Xuhui Zhou, Weihua Du, Xingyao Wang, Sean Welleck, Graham Neubig, Maarten Sap, and Yiming Yang. 2025. Training proactive and personalized llm agents. *arXiv preprint arXiv:2511.02208*.

Manan Suri, Puneet Mathur, Nedim Lipka, Franck Dernoncourt, Ryan A Rossi, and Dinesh Manocha. 2025. Structured uncertainty guided clarification for llm agents. *arXiv preprint arXiv:2511.08798*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, and 1 others. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

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*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

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, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

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

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

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.

A Experimental Setup Details

A.1 Benchmarks

τ-bench. τ-bench (Yao et al., 2024) simulates realistic multi-turn dialogues between an LLM agent and a user simulator in domain-specific environments equipped with API tools and rules. It provides two domains: **Retail** (product inquiry, order management, returns) and **Airline** (flight booking, modification, cancellation with fare-class restrictions). Each task involves 3 to 8 interaction turns with the user simulator issuing follow-up requests conditioned on prior agent responses. Evaluation uses reward-based metrics: $Avg@k$ measures the mean task-completion reward over k independent trials, and $pass^k$ measures the probability that all k trials succeed.

VitaBench. VitaBench (He et al., 2025) is a life-service simulation benchmark requiring agents to complete complex, tool-intensive tasks across four sub-scenarios: **Delivery** (food ordering with address and time constraints), **In-store** (restaurant reservation and service inquiries), **OTA** (travel planning involving transportation, accommodation, and attraction coordination), and **Cross-domain** (tasks spanning multiple service categories with shared state). Tasks feature verbose tool outputs, multi-entity coordination, and progressively revealed user constraints. Evaluation uses rubric-based scoring from 0 to 100 assessed by a GPT-4.1 (OpenAI, 2025) judge.

UserBench. UserBench (Qian et al., 2025a) evaluates agents on realistic user interactions where goals are underspecified and incrementally revealed. Unlike benchmarks with fully specified task descriptions, UserBench requires agents to proactively clarify ambiguous requirements and elicit user preferences. It reports three metrics: *Score* (overall task completion quality), *CER* (Correct Exist Rate, measuring whether the agent correctly identifies existing information), and *PE* (Preference Elicited, measuring the agent's ability to actively elicit user preferences before acting).

A.2 Baselines

- **ReAct** (Yao et al., 2022) retains complete reasoning-action-observation traces without compression, serving as the standard agentic paradigm.
- **StateAct** (Rozanov and Rei, 2025) augments ReAct with chain-of-states, explicitly generating goal, state, thought, and action at each turn, but does not separate tool-grounded facts from task-state judgments.
- **ReSum** (Wu et al., 2025) periodically invokes an external summarizer to compress interaction history into compact reasoning states, controlling context growth at the cost of additional LLM calls.
- **IterResearch** (Chen et al., 2025) restructures the workspace into an evolving report, synthesizing new findings at each turn while discarding raw interaction traces.
- **U-Fold** (Su et al., 2026) dynamically generates intent-aware dialogue summaries and extracts compact, task-relevant tool logs at each turn through two LLM-based modules.

B Prompt Templates

B.1 System Prompt with IDSS State

System Prompt with IDSS State

```
You are a task-oriented assistant. You help users complete multi-step tasks by calling tools, gathering information, and taking actions on behalf of the user.

BEHAVIORAL GUIDELINES
- Always verify facts from the Fact Layer before making claims. Never fabricate entity attributes.
- If a required variable is unknown, ask the user (if askable) or call a tool (if retrievable).
- Before executing an action, check all associated constraints. Do not proceed if any constraint is in VIOLATED status.
- When an intent is BLOCKED, explain the reason to
  the user and propose feasible alternatives.
- After completing a sub-task, review remaining PENDING intents before ending the conversation.

CURRENT TASK STATE (IDSS)
The following is your structured understanding of the current task. Use it as your primary decision-making reference. The state is automatically maintained across turns. You must update it after each response.
=== FACT LAYER (verified entities) ===
{fact_layer_snapshot}
=== STATE LAYER (task understanding) ===
{state_layer_snapshot}

STATE MAINTENANCE PROTOCOL
For EVERY response, update the <state> tag
```

```
with the current state layer. Follow this protocol:
1. Intents: Update status of each intent:
  - ACTIVE → currently processing
  - PENDING → known but not yet started
  - COMPLETED → finished (record outcome)
  - BLOCKED → infeasible (record reason)
  Add new intents when the user introduces new goals.
2. Variables: For each information slot:
  - Mark source: ASKABLE / RETRIEVABLE / DERIVABLE
  - Update value and known status after each turn.
  - Add new variables as task requirements emerge.
3. Constraints: For each constraint:
  - SATISFIED → condition currently met
  - UNSATISFIED → not yet verifiable
  - VIOLATED → confirmed impossible
  When VIOLATED, mark the associated intent as BLOCKED with factual evidence.
```

B.2 State Tag Output Format

Below is a concrete example from a τ-bench airline scenario where the user requests a flight modification and two checked bags.

State Tag Example (τ-bench Airline Scenario)

```
<state>
intents:
- name: "modify_flight"
  status: blocked
  reason: "R123.cabin = basic_economy
      violates not_basic_economy"
  depends_on: []
- name: "cancel_and_rebook"
  status: active
  reason: "alternative enabled by blocked
      modification + cancel_within_24h"
  depends_on: ["modify_flight.blocked"]
- name: "add_checked_bags"
  status: pending
  depends_on:
    ["cancel_and_rebook.completed"]
variables:
- name: "reservation_id"
  value: "R123"
  source: askable | known: true
- name: "original_route"
  value: "SFO to JFK"
  source: retrievable | known: true
  evidence: "Reservation R123.route"
- name: "checked_bags"
  value: 2
  source: askable | known: true
- name: "replacement_flight"
  value: null
  source: retrievable | known: false
constraints:
- scope: "flight_modification"
  condition: "fare is not basic economy"
  status: violated
  evidence: "R123.cabin = basic_economy"
- scope: "alternative_booking"
  condition: "cancel within 24h, user confirmed"
  status: satisfied
  evidence: "within_24h = true,
      user_confirmation = yes"
- scope: "new_reservation"
  condition: "replacement supports
      checked bags"
  status: unsatisfied
  evidence: null
</state>
```

B.3 Fact Layer Entity Format

Below is an example from a τ-bench airline task.

Fact Layer Snapshot (τ-bench Airline Task)

```
[Entity: Reservation #RSV-20240315]
  type: reservation
  id: RSV-20240315
  source: get_reservation("RSV-20240315")
  passenger_name: "Alice Chen"
  flight_number: UA-1234
  date: 2024-03-20
  departure: SFO 09:30 → JFK 18:05
  cabin_class: economy
  booking_status: confirmed
  change_fee: $150
  payment_method: Visa ending 4392
  baggage_allowance: 1 x 23kg
  seat_assignment: 24A (window)
[Entity: Flight UA-5678]
  type: flight (search result)
  id: UA-5678
  source: search_flights({origin: "SFO", dest: "JFK",
      date: "2024-03-20", cabin: "economy"})
  route: SFO 07:15 → JFK 15:50
  cabin_class: economy
  seats_available: 12
  price_difference: +$45
  change_eligible: true
[Entity: Flight UA-9012]
  type: flight (search result)
  id: UA-9012
  source: search_flights({...})
  route: SFO 11:30 → JFK 20:15
  cabin_class: economy
  seats_available: 3
  price_difference: +$20
  change_eligible: true
```

C Examples

C.1 τ-bench Retail: Multi-Item Exchange with Conditional Preferences

Example 1: Multi-item exchange with conditional fallback preferences

Benchmark and task: τ-bench Retail, multi-item exchange with a conditional fallback preference.

User goal:

"You are Yusuf Rossi in 19122. You received order #W2378156 and wish to exchange the mechanical key-

board for one with clicky switches, and the smart thermostat for one compatible with Google Home instead of Apple HomeKit. If there is no clicky + RGB + full size keyboard, you'd go for no backlight."

Turn 1 [User]
"Hi, I received order #W2378156. I want to exchange two items: the mechanical keyboard, for which I'd like one with clicky switches, RGB backlight, full size. And the smart thermostat, for which I need Google Home compatible instead of Apple HomeKit."

Turn 2 [Tool Call] `find_user_id_by_name_zip`

```
{ first_name: "Yusuf", last_name: "Rossi",
zip: "19122" }
→ "yusuf_rossi_9620"
```

Turn 3 [Tool Call]
`get_order_details("#W2378156")`

```
{
order_id: "#W2378156",
status: "delivered",
items: [
{ name: "Headphones", item_id:
"4202497723",
price: 342.81, options: {type:
"over-ear", connectivity: "wireless"}
},
{ name: "Vacuum Cleaner", item_id:
"4602305039", price: 561.05 },
{ name: "Mechanical Keyboard", item_id:
"1151293680",
product_id: "1656367028",
price: 272.33,
options: {switch_type: "linear",
backlight: "RGB", size: "full size"}
},
{ name: "Smart Thermostat", item_id:
"4983901480",
product_id: "4896585277",
price: 262.47,
options: {compatibility: "Apple HomeKit",
color: "black"} },
{ name: "Smart Watch", item_id:
"9408160950", price: 381.26 }
],
payment_method: "credit_card_9513926"
}
```

Turn 4 [Tool Call]
`get_product_details("1656367028")`

```
{ name: "Mechanical Keyboard", variants:
[
{ item_id: "9690244451",
options: {switch: "clicky", backlight:
"RGB", size: "60%"},
available: false },
{ item_id: "7706410293",
options: {switch: "clicky", backlight:
"none", size: "full size"},
available: true, price: 269.16 },
{ item_id: "6342039236",
options: {switch: "clicky", backlight:
"white", size: "full size"},
available: true, price: 244.91 },
...
] }
```

Turn 5 [Tool Call]
`get_product_details("4896585277")`

```
{ name: "Smart Thermostat", variants: [
...
{ item_id: "7747408585",
options: {compatibility: "Google
Assistant", color: "black"},
available: true, price: 249.01 },
{ item_id: "8722653925",
options: {compatibility: "Google
Assistant", color: "white"},
available: false },
...
] }
```

```
IDSS State (After Turn 5)
Fact Layer:
[Entity: Order] id: #W2378156
source: get_order_details()
status: delivered, payment: "credit_card_9513926"
[Entity: Product] id: 1656367028 (Keyboard)
source: get_product_details()
variants: [9690244451 (clicky+RGB+60%, unavail.),
7706410293 (clicky+none+full, $269.16)]
[Entity: Product] id: 4896585277 (Thermostat)
source: get_product_details()
variants: [7747408585 (Google+black, $249.01)]
State Layer:
Intents:
I1: "Exchange keyboard" → blocked
reason: primary variant unavailable
I1-fallback: "No backlight" → active
depends_on: [I1.blocked]
I2: "Exchange thermostat" → active
Variables:
  payment_method: "credit_card_9513926"
    source: derivable | known: true
    evidence: "order #W2378156.payment"
  price_diff_keyboard: -$3.17
    source: derivable | known: true
    evidence: "269.16 – 272.33"
  price_diff_thermostat: -$13.46
    source: derivable | known: true
    evidence: "249.01 – 262.47"
Constraints:
  C1: "clicky+RGB+full available" → violated
    evidence: "9690244451 unavailable"
  C2: "fallback: no backlight" → satisfied
    evidence: "C1 violated, user fallback"
  C3: "Google Home variant" → satisfied
    evidence: "7747408585 available"
Next Action:
  Confirm both exchanges before calling API.
```

Turn 6 [IDSS Agent]
"I checked the options. The clicky + RGB + full size keyboard is out of stock. Per your fallback preference, I'll use the clicky + no backlight + full size variant ($269.16).
For the thermostat, the Google Assistant model in black is available ($249.01). Shall I proceed with both exchanges?"

Turn 7 [User] *"Yes, go ahead."*

Turn 8 [Tool Call]
`exchange_delivered_order_items`

```
{
order_id: "#W2378156",
item_ids: ["1151293680", "4983901480"],
new_item_ids: ["7706410293", "7747408585"],
payment_method_id: "credit_card_9513926"
}
→ success
```

ReAct Failure Mode

ReAct attempts to exchange for the clicky+RGB+full variant (item 9690244451) without checking availability, causing an API error. After retrying, it **loses track of the conditional fallback** ("if no RGB, then no backlight") and either:

- selects a wrong variant (e.g., white backlight instead of none), or
- only exchanges one of the two items, forgetting the thermostat.

Result: 2 to 3 additional correction turns required, task only partially completed.

C.2 VitaBench Cross-domain: CS Trip with Schedule Conflict and Delivery Constraint

Example 2: Cascading time constraints across train, CS booking, and delivery

Benchmark and task: VitaBench Cross-domain, schedule repair across transportation, attraction booking, and delivery.

User goal:
User U624274 (female engineer, age 30–35, Baoding) and 5 friends plan a real-person CS activity in Beijing on Saturday. Existing orders: train Baoding→Beijing (G505, Sat), return train (Sun), CS tickets (6 adult, Sat 14:00). Company announces a Saturday morning meeting, requiring later departure and cascading rescheduling.

Turn 1 [User]
"My company scheduled a meeting Saturday morning. I need a high-speed rail departing after 1:00 PM; please ask my friend to rebook. The CS activity needs to be pushed back. Allow about 1.5h travel from station to CS base."

Turn 2 [Tool Call] `search_trains`

```
search_trains({from: "Baoding", to: "Beijing",
date: "2025-06-21", depart_after: "13:00"})
→ [
{ train_id: "T00012",
number: "G661",
depart: "13:28", arrive: "14:15",
price: 95, seat_type: "Second class" },
{ train_id: "T00014",
number: "G515",
depart: "14:02", arrive: "14:48",
price: 92, seat_type: "Second class" },
...
]
```

Turn 3 [Tool Call]
`get_order("S17550802154486284_O00004")`

```
{
order_type: "attraction",
store: "Beijing Tactical CS Base, Huilongguan, Changping District",
status: "paid",
products: [{ name: "Adult ticket", price: 398, qty: 6,
date: "2025-06-21" }],
reservation_time: "14:00",
location: { lat: 40.098765, lng: 116.345678 }
}
```

IDSS State (After Turn 3)

Fact Layer:
[Entity: Train] id: G661
source: search_trains()
depart: 13:28, arrive: 14:15, price: 95
[Entity: Train] id: G515
source: search_trains()
depart: 14:02, arrive: 14:48, price: 92
[Entity: Reservation] id: CS_O00004
source: get_order()
venue: "Beijing Tactical CS Base"
reservation_time: 14:00, pax: 6, status: paid

State Layer:
Intents:
I1: "Select train after 13:00" → **active**
I2: "Reschedule CS reservation" → **pending**
depends_on: [I1.resolved]
I3: "Notify friend" → **pending**
depends_on: [I1.resolved]
Variables:
departure_constraint: "after 13:00"
source: askable | known: true
travel_time_to_cs: "1.5h"
source: askable | known: true
selected_train: null
source: derivable | known: false
new_cs_time: null
source: derivable | known: false
Constraints:
C1: "depart ≥ 13:00" → **satisfied**
evidence: "G661=13:28, G515=14:02"
C2: "arrive+travel ≤ CS start" → **violated**
evidence: "14:15+1.5h=15:45 > 14:00"
C3: "CS starts before 17:00" → **unsatisfied**
evidence: "new_cs_time unknown"
Next Action:
Recommend G661 and ask for confirmation.

Turn 4 [IDSS Agent]
"I found G661 departing at 13:28 (arriving 14:15). With 1.5h travel to the CS base, you'd arrive around 15:45. I'll reschedule the CS reservation from 14:00 to 15:30 for a buffer. G515 departs at 14:02, pushing CS start to 16:18—possibly too late for a full afternoon session. Shall I book G661 and update the reservation to 15:30?"

Turn 5 [User]
"Yes, G661 is good. Also, I need sunscreen before the trip. Check if Anessa can be delivered to my home today before 10 PM."

Turn 6 [Tool Call] `search_delivery`

```
search_delivery({
product: "Anessa sunscreen",
address: "Room 502, Unit 2, Building 8,
Oriental Garden, Baoding"
})
→ {
available: true,
product: "Shiseido Anessa SPF50+ PA++++
90ml",
price: 249,
store: "JD Beauty Supermarket",
delivery_slot: "today 20:00 to 22:00"
}
```

```
IDSS State (After Turn 6)
Fact Layer:
[Entity: Delivery] id: pending_order
source: search_delivery()
product: "Anessa SPF50+ 90ml", price: 249
store: "JD Beauty Supermarket"
slot: today 20:00-22:00
State Layer:
Intents:
I1: "Book train G661" → active
I2: "Reschedule CS to 15:30" → active
I3: "Notify friend" → pending
I4: "Order sunscreen delivery" → active
Variables:
  delivery_deadline: "before 22:00"
    source: askable | known: true
  quiet_delivery: null
    source: askable | known: false
    evidence: "slot 20:00-22:00, late
evening"
Constraints:
  C4: "delivery ≤ 22:00" → satisfied
    evidence: "slot ends 22:00"
  C5: "quiet delivery" → unsatisfied
    evidence: "slot 20:00-22:00"
Next Action:
  Confirm quiet-delivery, then place
order.
```

Turn 7 [IDSS Agent]
"Anessa SPF50+ 90ml (¥249) can be delivered today between 8 and 10 PM. Since it's a late evening delivery, would you like me to add a note to knock quietly or leave at door?"

ReAct Failure Mode

ReAct changes the train ticket but **fails to propagate the time constraint** to the CS reservation (the dependency chain `train_arrival` → `travel_time` → `CS_start` is not tracked). Specifically:

- Books G661 but leaves CS reservation at 14:00 (impossible to attend).
- Orders sunscreen without verifying the delivery time constraint.
- After 3 additional user correction turns, plan still has timing inconsistency.

C.3 UserBench: Business Trip with Underspecified Preferences

Example 3: Eliciting implicit preferences for flights and restaurants

Benchmark and task: UserBench TravelGym, underspecified flight and restaurant preferences.

User goal:

"I am planning a business trip to NYC from December 10th to 15th. I want my flight from LA on December 10th to include carry-on baggage allowance [...] Additionally, I need generous checked luggage service because I plan to bring back souvenirs. During my stay, dining in restaurants with a conducive environment for business discussions is essential. I usually avoid places with any 1- or 2-star reviews."

Hidden preferences (ground truth, not revealed upfront):
• *Flight*: carry-on baggage allowance + checked bags service
• *Restaurant*: all reviews ≥ 3-star + "Business Dining" tag

Turn 1 [User]
"I need to book a flight from LA to NYC on Dec 10th and find two good restaurants for business dinners."

Turn 2 [IDSS Agent]
"I'd be happy to help. A few questions: (1) Do you have luggage preferences—carry-on only, or do you need checked bags? (2) For the restaurants, any cuisine preference? Is the ambiance important—casual, upscale, or specifically suited for business meetings?"

Turn 3 [User]
"I always keep travel essentials with me for unexpected layovers, and I'm bringing back souvenirs so I'll need checked luggage too. For restaurants, the environment matters because the conversation might touch on business."

```
IDSS State (After Turn 3)
Fact Layer: (empty – no tool calls yet)
State Layer:
Intents:
I1: "Search flights" → active
reason: flight preferences are known
I2: "Elicit rating preference" → active
reason: rating threshold unknown
I3: "Search restaurants" → pending
depends_on: [I2.completed]
Variables:
  origin: "LAX"
    source: askable | known: true
  destination: "NYC"
    source: askable | known: true
  date: "Dec 10"
    source: askable | known: true
  carry_on: "required"
    source: derivable | known: true
    evidence: "keep travel essentials
with me"
  checked_bags: "required"
    source: derivable | known: true
    evidence: "bringing back souvenirs"
  restaurant_ambiance: "business dining"
    source: derivable | known: true
    evidence: "conversation touches
business"
  rating_threshold: null
    source: askable | known: false
Constraints:
  C1: "carry-on preference known" →
satisfied
    evidence: "carry_on = required"
  C2: "checked bags preference known" →
satisfied
    evidence: "checked_bags = required"
  C3: "dining ambiance known" →
satisfied
    evidence: "ambiance = business
dining"
  C4: "rating threshold known" →
unsatisfied
    evidence: "rating_threshold = null"
Next Action:
  Ask about restaurant rating tolerance.
```

Turn 4 [IDSS Agent]
"Got it—I'll filter for flights with both carry-on and checked bag service. One more thing about restaurants: do you have a minimum quality bar? For example, do you mind if a place has some lower ratings, or do you prefer consistently high reviews?"

Turn 5 [User]
"I usually scroll past restaurants with any 1- or 2-star reviews. I don't want to take the risk, especially for a business dinner."

```
IDSS State (After Turn 5)
Fact Layer: (empty – no tool calls yet)
State Layer:
Intents:
I1: "Search flights" → active
I3: "Search restaurants" → active
depends_on: [I2.completed]
Variables:
  carry_on: "required"
    source: derivable | known: true
    evidence: "keep travel essentials
with me"
  checked_bags: "required"
    source: derivable | known: true
    evidence: "bringing back souvenirs"
  restaurant_ambiance: "business dining"
    source: derivable | known: true
    evidence: "conversation touches
business"
  rating_threshold: "all reviews ≥
3-star"
    source: askable | known: true
    evidence: "avoids 1-2 star reviews"
Constraints:
  C1: "includes carry-on" → unsatisfied
  C2: "supports checked bags" →
unsatisfied
  C3: "Business Dining tag" →
unsatisfied
  C4: "no 1-2 star reviews" → unsatisfied
    (all pending tool verification)
Next Action:
  Search flights and restaurants.
```

Turn 6 [Tool Call] `interact_with_env` (flight search)

```
search("flight LAX to NYC Dec 10") → [
F8: { company: "Delta Airlines",
flight_number: "DL456",
cost: $350,
amenities: ["WiFi", "Carry on Baggage
Allowance"],
service: { checked_bag_cost: $50 },
duration: 5.5h },
F16: { company: "American Airlines",
flight_number: "AA789",
cost: $400,
amenities: ["WiFi", "Carry on Baggage
Allowance"],
service: { checked_bag_cost: $60 } },
F21: { company: "Southwest Airlines", ...
},
...
]
```

Turn 7 [Tool Call] `interact_with_env` (restaurant search)

```
search("restaurant NYC Dec 11") → [
R17: { name: "The Executive Table",
cuisine: "French",
rating: 9,
reviews: { 5⋆: 320, 4⋆: 180, 3⋆: 50,
2⋆: 0, 1⋆: 0 },
tags: ["Business Dining", "Reservations",
"Parking"] },
R8: { name: "Sushi Prestige",
cuisine: "Japanese",
rating: 8,
reviews: { ..., 2⋆: 0, 1⋆: 0 },
tags: ["Business Dining", "Walk-ins
Welcomed"] },
R21: { name: "Urban Grill",
reviews: { ..., 2⋆: 5, 1⋆: 2 },
tags: ["Casual Dining", "Late Night"] },
...
]
```

Turn 8 [IDSS Agent]

"Here are my recommendations:

Flight: *Delta DL456 ($350 + $50 checked bag), including carry-on allowance and checked bag service.*

Restaurants:

(1) The Executive Table (French, all reviews 3-star+, Business Dining tag)

(2) Sushi Prestige (Japanese, all reviews 3-star+, Business Dining tag)

I filtered out places like Urban Grill that have 1- to 2-star reviews. Shall I book these?"

IDSS State (After Turn 7)

Fact Layer:

```
[Entity: Flight] id: DL456
source: interact_with_env()
company: Delta, cost: $350
amenities: [WiFi, Carry on Baggage
Allowance]
checked_bag_cost: $50
[Entity: Restaurant] id: R17
source: interact_with_env()
name: "The Executive Table", cuisine:
French
min_review: 3⋆, tags: [Business Dining]
[Entity: Restaurant] id: R8
source: interact_with_env()
name: "Sushi Prestige", cuisine: Japanese
min_review: 3⋆, tags: [Business Dining]
[Entity: Restaurant] id: R21 — filtered
min_review: 1⋆ → violates C4
```

State Layer:

```
Intents:
I1: "Book flight" → completed
I3: "Recommend restaurants" → completed
Variables:
  carry_on: "required"
    source: derivable | known: true
  checked_bags: "required"
    source: derivable | known: true
  rating_threshold: "all reviews ≥
3-star"
    source: askable | known: true
Constraints:
  C1: "includes carry-on" → satisfied
    evidence: "DL456 amenities"
  C2: "supports checked bags" →
satisfied
    evidence: "DL456
checked_bag_cost=$50"
  C3: "Business Dining tag" → satisfied
    evidence: "R17, R8 tagged"
  C4: "no 1-2 star reviews" → satisfied
    evidence: "R17 min=3 star, R8 min=3
star;
    R21 has 2 star → excluded"
Next Action:
  Present recommendations, await
confirmation.
```

ReAct Failure Mode

ReAct immediately searches flights without asking about baggage needs:

- Selects cheapest flight **without checked bag service** (user preference missed).
- For restaurants, picks highest-rated option without filtering for "no 1–2 star reviews" or "Business Dining" tag; recommends Urban Grill (has 2-star reviews).
- **Never elicits** user's implicit preferences about baggage or review threshold.

Result: Low Preference Elicitation (PE) score. The user receives misaligned recommendations.