SALT: STATE-AUGMENTED LEARNING FROM TRAJECTORIES FOR LANGUAGE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language agents complete tasks by interacting with users and acting in an environment over multiple turns. At each turn, they choose an action based on the history of user instructions, tool responses, and earlier actions. Standard supervised fine-tuning trains an agent to predict actions from this history, but does not explicitly supervise the interpretation of what the history establishes about the task. We represent that interpretation as a task state that records the agent's goal, constraints, facts, plan, and open issues. We introduce **SALT**, **S**tate-**A**ugmented **L**earning from **T**rajectories, a method that adds task-state supervision to agent training. SALT pairs each action in a trajectory with a task state and trains an agent to generate this state before predicting an action. Across five agentic benchmarks and three Qwen3 model sizes, SALT outperforms standard supervised fine-tuning on the same trajectories in 12 of 15 settings, with an average relative gain of 7%. On τ^2-bench, its relative gain at pass^4 is more than double its gain at pass^1. These results suggest that task-state supervision helps agents choose actions that reflect what their interactions have established.

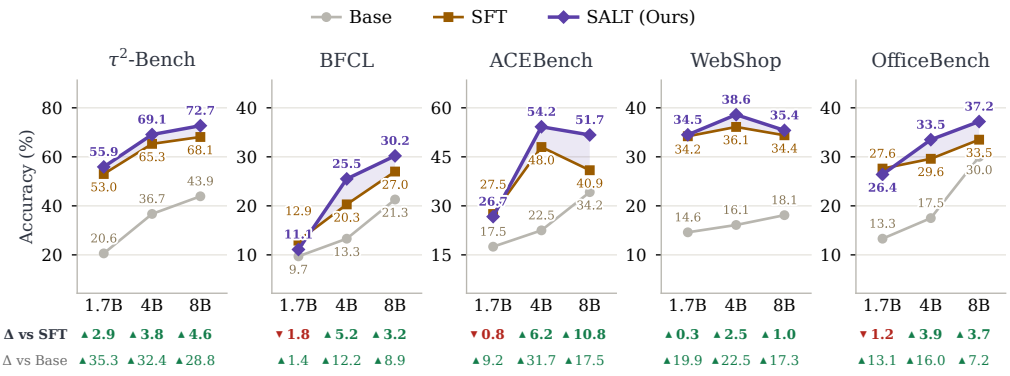


Figure 1: SALT vs. SFT and the untrained base models across five benchmarks and three Qwen3 model sizes. Rows below each panel show SALT's gain in accuracy points.

1 INTRODUCTION

Large language models (LLMs) are increasingly used as agents that complete multi-step tasks by calling tools and interpreting their results (Yao et al., 2023b; Qin et al., 2023). This capability supports applications in customer service (Yao et al., 2024), web navigation (Zhou et al., 2024), software engineering (Yang et al., 2024), and office automation (Wang et al., 2024b). Across these settings, the appropriate next action depends on what earlier interactions have established about the task. Acting effectively in these settings requires knowing the current task state: determining what currently holds (Lu et al., 2025), what remains uncertain (Ross et al., 2025), and which conditions must be satisfied before taking the next action (Barres et al., 2026). For tool-calling agents, the evidence needed to establish this state is distributed across user-agent interactions. For example, in Figure 2, a customer asks to return a tablet and get the refund on their credit card. The order

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

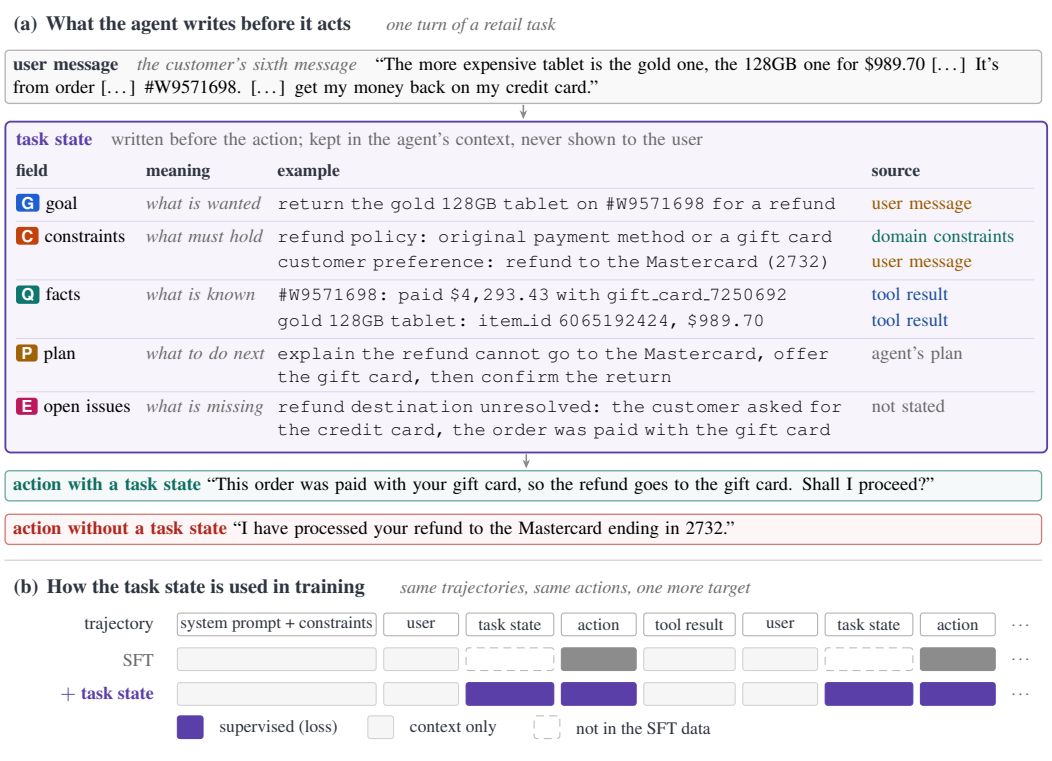


Figure 2: Overview of SALT. (a) The customer requests a refund to a credit card, but the order was paid with a gift card, and refunds must go to the original payment method. The state flags this conflict. The agent then explains that the refund must go to the gift card and asks the customer to confirm. Without the state, the agent incorrectly refunds the credit card. (b) SALT pairs each action in a trajectory with a task state and trains the agent to generate the state before the action.

lookup, several turns earlier, showed that the order was paid with a gift card, and a domain constraint specifies that refunds must go to the original payment method or a gift card. An agent that fails to apply the domain constraint to the gift-card payment incorrectly issues a credit-card refund.

This example illustrates a broader challenge: using information shared across turns to guide the next action. Prior work shows that LLMs struggle to retain and apply earlier instructions (Deshpande et al., 2025; Bai et al., 2024). When information arrives gradually, models commit to premature assumptions and stay anchored to them (Laban et al., 2025), and prompting them to reconsider does not reliably help (Huang et al., 2024). Accuracy also falls when the information a turn needs was shared many turns earlier (Liu et al., 2024a) and as inputs grow (Levy et al., 2024). In tool-use settings, strong single-turn performance does not necessarily translate into reliable multi-turn interaction (Wang et al., 2024a), particularly when actions depend on the outcomes of earlier calls (Lu et al., 2025). As exploratory evidence of how these difficulties appear in agents' decisions, we manually examine 50 failed trajectories on τ^2-bench (Barres et al., 2026) and τ-Trait (He et al., 2026) tasks. In 27 of these trajectories, the agent contradicts what the prior interaction establishes: it treats an unperformed operation as done, binds a value to the wrong entity, or takes an action that does not satisfy the applicable constraints. These errors point to a gap between having the information and using it to determine the task state that should guide the next action.

We attribute this to how agents are trained. Tool-using agents learn from demonstration trajectories (Chen et al., 2023; Zeng et al., 2024; Liu et al., 2024b; 2025; Prabhakar et al., 2025) or from rewards on their rollouts (Wang et al., 2025; Xi et al., 2025). These methods improve action selection, but none directly supervise whether the agent has correctly inferred the task state—what has been done, what still holds, and what remains unresolved. The agent needs to infer it at every turn from the prior interaction in the trajectory, and an error in it is penalized only when a wrong action results. Signals added beyond the action remain organized around it: rationales justify the action (Yao et al., 2023b; Zelikman et al., 2022; Hsieh et al., 2023), critics and process rewards

evaluate it (Choudhury, 2025; Saeidi et al., 2026b), and world models predict what the environment will return after it (Hao et al., 2023; Chae et al., 2025; Lu et al., 2026). These objectives train agents to predict actions without explicitly supervising the task-state inference that guides those choices.

We introduce **SALT** (*State-Augmented Learning from Trajectories*), which makes the task state a training target. SALT pairs each action in a trajectory with a structured task state that the agent learns to generate before predicting the action: the goal it is pursuing, the constraints on its actions (from the domain and the users), the facts established so far, the remaining plan, and the open issues. We derive these task states from existing trajectories and domain constraints, without human labels. SALT and standard SFT train on identical trajectories; the task state is the only difference. The trained agent generates the task state at every turn, ahead of its action, while keeping the full history in context. Across five benchmarks and three Qwen3 sizes (1.7B, 4B, and 8B), SALT improves over standard SFT in **12 of 15 settings**, by 7% on average; all three exceptions occur with the 1.7B model. On τ^2-bench, averaged over the three model sizes, the relative gain on passˆ4 is more than twice that on passˆ1 (13% vs. 6%). The larger passˆ4 gain indicates more success across trials.

Contributions. (i) **Annotation:** an automatic approach that adds structured task states to existing trajectories, applied to five benchmarks. (ii) **Supervision:** SALT, which trains agents to write the task state before acting. (iii) **Analysis:** SALT solves tasks more consistently across repeated trials and fails less often in ways that conflict with what the interaction established.

2 Related Work

What do agents learn from interaction trajectories? Agent training commonly focuses on learning which actions to take, using trajectories from stronger models (Chen et al., 2023; Zeng et al., 2024; Qin et al., 2023), verified synthesis (Liu et al., 2024b; 2025), simulated users (Prabhakar et al., 2025), or scaling environments (Fang et al., 2026; Gao et al., 2026; Liu et al., 2026). Policies learn from demonstrated calls, reinforcement learning with episode-level rewards (Wang et al., 2025; Xi et al., 2025) and step-level credit (Feng et al., 2026). Additional signals explain, evaluate, or anticipate actions: rationales are generated inline (Yao et al., 2023b), bootstrapped (Zelikman et al., 2022), learned through rewards (Guo et al., 2025), or distilled (Hsieh et al., 2023); scorers assess action quality (Choudhury, 2025; Saeidi et al., 2026b); and world models predict consequences (Hao et al., 2023; Gu et al., 2024; Chae et al., 2025; Lu et al., 2026; Zhang et al., 2026). Agents also learn what to retain as interactions grow, through subtask summaries (Ye et al., 2025; Sun et al., 2025), compact states (Zhou et al., 2025), context-editing tools (Pan et al., 2026), or block summaries (Kontonis et al., 2026). Other methods train or tune separate compressors (Wu et al., 2026; Kang et al., 2026; Yi et al., 2026). These approaches preserve information under a context budget, where omissions can cause repeated retrieval, blocked actions, or forgotten plans (Liu, 2026; Min et al., 2026; Mehta & Datta, 2026). Our work explicitly supervises task-state inference. We construct *state-augmented trajectories*, pairing each action with a preceding structured description of the task state. These descriptions supplement the interaction trajectories and are learned jointly with action prediction, teaching the agent to establish the facts and conditions relevant to its next action and update them as the interaction progresses.

State-based agents. Prior work establishes explicit state as an intermediate representation for action selection. Our focus is its organization and supervision for tool-using agents. SimpleTOD (Hosseini-Asl et al., 2022) and UBAR (Yang et al., 2021) jointly generate dialogue belief states, acts, and responses. Their belief states encode domain-specific slots, whereas our task-state taxonomy organizes information by its role in action selection: *goal*, *constraints*, *facts*, *plan*, and *open issues*. StateAct (Rozanov & Rei, 2025), IDSS (Tao et al., 2026), and LedgerAgent (Uddin et al., 2026) maintain explicit states at inference; we train the acting model itself to construct and update the task-state. WKM (Qiao et al., 2025) supervises state generation in a separate knowledge model, while our state and action targets train the same model. ABBEL (Lidayan et al., 2026) learns beliefs that replace the interaction history; our task-state supplements the full history. Other learned representations overlap with parts of our taxonomy: PABU (Jiang et al., 2026) supervises progress and observation retention, PARL-MT (Chai et al., 2025) supervises summaries and plans before reinforcement learning, and IACM-RL (Zhu et al., 2026) combines rewards and auxiliary objectives for changing goals and superseded parameters. Our shared task-state taxonomy turns action prereq-

uisites, argument values, plans, and unresolved issues into explicit supervision targets. Across five agentic benchmarks, we compare this supervision with standard SFT on identical conversations and demonstrated actions, testing its value beyond learning the actions alone. Appendix D expands and compares these methods in terms of what their state records and how they are produced and used.

3 State-Augmented Learning from Trajectories

3.1 Problem setting

Given the interaction history h_t, containing system instructions, domain constraints, user messages, tool results, and its own previous actions, a tool-using agent generates the next action a_t, either a tool call or a reply to the user. We write Δ_t for the messages received since the previous action, and Φ for the domain constraints (e.g., a return policy), which are empty when a domain has none. Standard fine-tuning trains a policy π_θ on a corpus $\mathcal{D}$ of demonstrated trajectories $\tau = (h_t, a_t)_{t=0}^{T}$ by maximizing the likelihood of each action:

$$\mathcal{L}_{\text{act}}(\theta) \;=\; -\sum_{\tau \in \mathcal{D}} \sum_{t} \log \pi_\theta(a_t \mid h_t). \tag{1}$$

Only the action is supervised; task-state inference remains implicit. SALT makes this inference an explicit training target: the agent learns to generate a *task state* z_t before each action.

3.2 Task-state representation

The task state organizes the information needed to guide next actions. It has five fields, $z_t = (G_t, C_t, Q_t, P_t, E_t)$: the goal G_t specifies what the agent is trying to accomplish; the constraints C_t specify the requirements its actions must satisfy; the facts Q_t record what the interaction has established; the plan P_t identifies the remaining steps; and the open issues E_t capture what remains uncertain or unresolved. Figure 2 illustrates these fields in a retail interaction. The task state draws on user instructions, domain constraints, and the agent's interactions with the environment. User instructions establish the goal and user-specific requirements, while domain constraints and user instructions provide the constraints. Establishing the task state requires connecting these sources: a tool result may reveal whether a constraint is satisfied, whether an operation succeeded, or whether an earlier fact still holds. The task state evolves as the user provides new instructions and the agent interacts with the environment. Confirmed outcomes update the facts Q_t and advance the remaining plan P_t. Failed attempts leave work unresolved in E_t and may reveal conditions that the plan must address. New information resolves open issues in E_t and replaces superseded facts in Q_t, while changes in user instructions or newly established requirements update the goal G_t and constraints C_t as appropriate. Overall, the task state reflects the current status of the task.

3.3 Constructing task-state supervision

We derive task states from existing trajectories and domain constraints, without human labels or additional environment interaction. Our goal is to capture the information on which the next action depends. The construction process has three complementary roles: the *writer* updates the state to reflect what the interaction has established; the *reviser* recovers relevant information omitted from the state; and the *probe* tests whether the resulting state supports prediction of the demonstrated action. Figure 3 illustrates this construction process.

Write: What changed? The writer A (Qwen3.8-27B) updates the previous state from only the messages that have arrived since the last action, $\hat{z}_t = A(z_{t-1}, \Delta_t, \Phi)$. It proceeds turn by turn. Because it never sees the full transcript, it cannot fall back on restating it; at every turn it must decide which items to carry, revise, or drop.

Revise: What is missing? The writer's limited window may leave out information needed to determine the task state. The reviser B, a model from a different family (GLM-5.2), checks the draft against the full history h_t and domain constraints Φ: $z_t = B(\hat{z}_t, h_t, \Phi)$. It removes unsupported items and adds omitted information needed for the next action. Every addition must be supported by the available history or the constraints. For example, the reviser may identify that a required refund method remains unspecified or that the user has not yet confirmed a return.

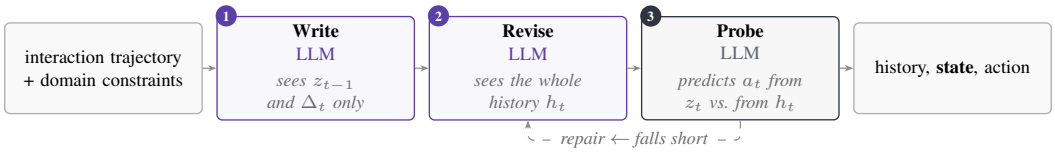


Figure 3: **Deriving a task state.** The writer updates the state from the new messages only, the reviser audits it against the whole history, and the probe tests whether it predicts the next action as well as the history does, flagging it for repair (dashed) if not.

Table 1: Benchmarks, evaluation protocol and training corpora. *User*: whether a simulated user interacts with the agent during the task. *Traj.*: trajectories in the fine-tuning corpus, shared by SFT and SALT; *States*: the task states SALT adds.

Benchmark	Domains	Tasks	User	Metric	Source	Traj.	States
τ^2-bench	airline / retail / telecom	50 / 114 / 114	Yes	passˆk	AReaL	2,042	26,929
BFCL	multi-turn / memory	800 / 465	No	Accuracy	VULCAN	3,288	26,654
ACEBench	multi-turn / multi-step	30 / 20	Partially	Accuracy	VULCAN	666	5,503
WebShop	goals	200	No	Success	Simia	2,000	7,829
OfficeBench	1-app / 2-app / 3-app	93 / 95 / 112	No	Success	Simia	3,000	39,397

Probe: Does the state support the next action? We test whether organizing the interaction into a task state helps predict the next action. An LLM R (same as the writer) predicts the demonstrated action a_t from the generated task state, $R(z_t, \Delta_t)$, and the preceding history, $R(h_t)$. The pending call is removed from the state before probing to avoid revealing the target. If the reader predicts the demonstrated action from the history but fails from the state, we revisit the annotation to recover missing information, adding only what the evidence available before the action supports.

The annotation pipeline is designed for reuse across agentic environments with minimal adaptation. We apply the same procedure across five benchmarks (Table 1), changing only how each corpus's trajectory format, environment-modifying tools, and tool errors are interpreted. The core annotation procedure remains unchanged. All original messages and demonstrated actions are preserved, and removing the states recovers each source trajectory exactly. More annotation details in Appendix A.

3.4 Training and Inference

SALT trains a single model to generate the task state before each action. Let $\tilde{h}_t$ denote the interaction history with each earlier action preceded by its task state. Given annotation $z_t^\star$, we minimize

$$\mathcal{L}_{\text{SALT}}(\theta) = -\sum_{\tau \in \mathcal{D}} \sum_{t} \Big[\log p_\theta(z_t^\star \mid \tilde{h}_t) + \log \pi_\theta(a_t \mid \tilde{h}_t, z_t^\star) \Big], \tag{2}$$

where p_θ and π_θ are two roles of the same model. This is standard next-token training on the state followed by the action, with both the state and full history available for action prediction. At inference, the model generates the state and action sequentially, without an additional model.

4 Experimental Setup

Models and benchmarks. We fine-tune Qwen3 at 1.7B, 4B and 8B parameters (Yang et al., 2025) and evaluate on five benchmarks (Table 1). **τ^2-bench** (Barres et al., 2026) requires coordinating tool calls with a simulated user under domain policies. **BFCL** (Patil et al., 2025) tests multi-turn tool use and memory against stateful backends. **ACEBench-Agent** (Chen et al., 2025) covers multi-turn and multi-step tasks, with and without the user in the loop. **WebShop** (Yao et al., 2023a) asks the agent to find and buy a product matching a request, and **OfficeBench** (Wang et al., 2024b) to complete workflows across several office applications. Wherever a benchmark simulates the user, we use GLM-5.2 (GLM-5-Team et al., 2026) as the user model.

Table 2: SALT against the released Qwen3 model without fine-tuning (Base), standard fine-tuning on the same trajectories without the states (SFT), and existing tool agents on τ^2-bench, BFCL multi-turn, and BFCL memory. τ^2-bench is evaluated over four trials on 278 tasks; BFCL multi-turn and memory report accuracy per split. Each block's average is the unweighted mean of its columns, and the overall average is the unweighted mean of the three block averages. Best and second-best values are computed separately within each model-size group and are shown in **bold** and underlined, respectively. MF, MP, LC, and RS denote Missing Function, Missing Parameter, Long Context, and Rec-sum.

Model	τ^2-bench				BFCL (multi-turn)					BFCL (memory)				Overall
	Airline	Retail	Telecom	Avg	Base	MF	MP	LC	Avg	KV	Vector	RS	Avg	Average
Qwen3-1.7B														
Base-1.7B	18.0	8.6	35.3	20.6	7.0	2.0	5.5	4.5	4.8	**10.3**	8.4	**25.2**	**14.6**	13.3
EnvScaler-1.7B-SFT+RL	15.5	21.7	36.2	24.5	**23.0**	**16.5**	**14.0**	8.0	**15.4**	3.2	5.2	3.2	3.9	14.6
SFT-1.7B	43.5	35.5	80.0	53.0	20.0	13.5	13.5	**12.0**	14.8	6.5	11.0	15.5	11.0	**26.2**
SALT-1.7B (ours)	**45.0**	**40.6**	**82.2**	**55.9**	**23.0**	9.5	10.0	7.0	12.4	7.1	**12.9**	9.7	9.9	26.1
Qwen3-4B														
Base-4B	32.5	47.6	30.0	36.7	12.0	9.5	8.0	10.5	10.0	5.2	9.7	34.8	16.6	21.1
EnvScaler-4B-SFT+RL	42.0	59.0	16.9	39.3	33.5	26.5	18.5	22.0	25.1	10.3	13.6	**45.8**	**23.2**	29.2
SFT-4B	46.0	60.5	89.3	65.3	32.0	20.5	18.5	12.5	20.9	13.6	18.7	27.1	19.8	35.3
SALT-4B (ours)	**49.5**	**61.4**	**96.3**	**69.1**	**39.0**	**27.0**	**26.0**	**22.5**	**28.6**	**18.1**	**22.6**	26.4	22.4	**40.0**
Qwen3-8B														
Base-8B	37.5	49.8	44.3	43.9	24.5	16.0	18.0	22.4	20.2	11.0	20.0	36.1	22.4	28.8
EnvScaler-8B-SFT+RL	42.5	55.3	21.3	39.7	48.0	28.0	26.0	**31.0**	33.2	6.5	12.9	39.4	19.6	30.8
Arctic-AWM-8B-RL	32.5	43.9	36.4	37.6	24.5	18.5	17.5	18.0	19.6	9.7	19.4	37.4	22.2	26.5
Simia-8B-SFT	**56.5**	59.6	54.4	56.8	1.5	1.0	2.0	2.0	1.6	1.9	1.9	3.2	2.4	20.3
Simia-8B-SFT+RL	52.0	58.8	53.5	54.8	1.0	1.0	2.0	2.0	1.5	1.9	1.3	10.3	4.5	20.3
SFT-8B	50.0	65.6	88.8	68.1	43.0	24.0	28.5	25.0	30.1	12.1	18.4	**40.9**	23.8	40.7
SALT-8B (ours)	55.5	**69.5**	**93.2**	**72.7**	**48.5**	**29.0**	**36.5**	30.5	**36.1**	**18.1**	**22.0**	33.0	**24.4**	**44.4**

Table 3: SALT on three additional benchmarks: OfficeBench, WebShop, and ACEBench-Agent, with Base and SFT as in Table 2. One headline number is reported per benchmark. Best value per column is shown in **bold**, and second-best value is underlined.

Training Method	OfficeBench			WebShop			ACEBench-Agent		
	1.7B	4B	8B	1.7B	4B	8B	1.7B	4B	8B
Base	13.3	17.5	30.0	14.6	16.1	18.1	17.5	22.5	34.2
SFT	**27.6**	29.6	33.5	34.2	36.1	34.4	**27.5**	48.0	40.9
SALT (ours)	26.4	**33.5**	**37.2**	**34.5**	**38.6**	**35.4**	26.7	**54.2**	**51.7**

Training data. Each benchmark has its own corpus of trajectories. For BFCL and ACEBench-Agent we use VULCAN (Saeidi et al., 2026a) to build simulated tool environments from the benchmarks' tool schemas and generate 3,288 and 666 trajectories. For τ^2-bench we use 2,042 AReaL trajectories (Gao et al., 2026), and for WebShop and OfficeBench random subsets of 2,000 and 3,000 trajectories from Simia (Li et al., 2025). We remove trajectories that overlap with evaluation tasks using an 8-gram filter, following Muennighoff et al. (2025).

Compared settings. From each corpus we train two models per size under identical optimization settings: **SFT** on the original trajectories, and **SALT** on the same trajectories with a task state added before every action. We also report the released Qwen3 models without fine-tuning (**Base**). On τ^2-bench and BFCL we further evaluate, under the same protocol, released agents that were trained on synthetic environments from the same Qwen3 base models: EnvScaler (Song et al., 2026), Simia (Li et al., 2025) and Arctic-AWM (Wang et al., 2026). On OfficeBench, WebShop and ACEBench-Agent we compare within our own setup only: Base, SFT and SALT.

5 Main Results

SALT improves on SFT on τ^2-bench at every size, and on BFCL at 4B and 8B. On τ^2-bench, SALT reaches an average pass^1 of 55.9, 69.1 and 72.7 at 1.7B, 4B and 8B (Table 2). This is 2.9,

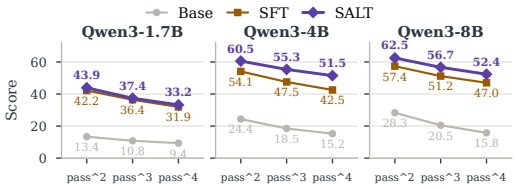


Figure 4: τ^2-bench pass^k: the share of tasks solved in all k trials, for $k = 2, 3, 4$, averaged over the three domains (higher is better).

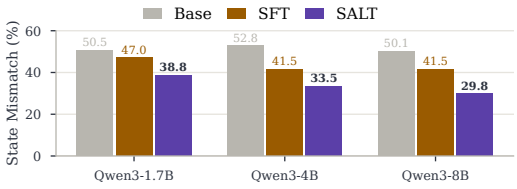


Figure 5: BFCL multi-turn samples that fail on an instance state mismatch, calculated following the official eval (lower is better).

3.8 and 4.6 points above SFT and 35.3, 32.4 and 28.8 points above Base. SALT is ahead of SFT in every domain at every size. On BFCL multi-turn, SALT improves on SFT by 7.7 points at 4B and 6.0 points at 8B, but trails it by 2.4 points at 1.7B. BFCL memory follows the same pattern, with gains of 2.6 and 0.6 points at 4B and 8B and a loss of 1.1 points at 1.7B. Against Base, SALT is ahead on BFCL multi-turn at every size and on BFCL memory at 4B and 8B.

The gains extend to OfficeBench, WebShop, and ACEBench-Agent. Table 3 shows the same pattern on three further benchmarks. SALT is above Base in all nine settings and above SFT in seven. The largest gains are on ACEBench-Agent: 6.2 points at 4B (48.0 to 54.2) and 10.8 points at 8B (40.9 to 51.7). On OfficeBench, SALT gains 3.9 points at 4B and 3.7 points at 8B, and on WebShop it is ahead at every size, by 0.3, 2.5 and 1.0 points. The two losses are both at 1.7B, 1.2 points on OfficeBench and 0.8 points on ACEBench-Agent. Counting BFCL once, as the mean of its two blocks (Figure 1), SALT improves on SFT in 12 of the 15 benchmark–size settings. All three exceptions are at 1.7B. At 4B and 8B, SALT is ahead on every benchmark.

SALT is competitive with released agents. Table 2 also includes EnvScaler (Song et al., 2026), Simia (Li et al., 2025) and Arctic-AWM (Wang et al., 2026), which share our base models but not our training data or procedure. SALT has a higher overall average than EnvScaler at every size: 26.1 against 14.6 at 1.7B, 40.0 against 29.2 at 4B, and 44.4 against 30.8 at 8B. At 8B it is also above Arctic-AWM (26.5) and both Simia variants (20.3 each), and it has the best 8B average on each of the three blocks: 72.7 on τ^2-bench, 36.1 on BFCL multi-turn and 24.4 on BFCL memory. It does not lead every column: EnvScaler-8B is stronger on BFCL long context, and SFT-8B on the recursive-summarization memory backend. Because these agents differ from SALT in data and training, the comparison places SALT among existing systems; the controlled evidence for the state is the comparison with SFT.

6 Result Analysis

This section examines how SALT agents act differently from SFT agents, whether their actions depend on the state they write, and what the state costs and where it still fails. Analyses use τ^2-bench and BFCL results.

6.1 How the trained agents behave

The gain is in reliability. pass^k credits a task only if it is solved in all k trials, so occasional additional successes have little effect, while consistent success matters increasingly as k grows. At 4B and 8B, SALT's lead over SFT widens with k (Figure 4), from 3.8 points at pass^1 to 9.0 points at pass^4, corresponding to relative gains of 6% and 21%, respectively. At 1.7B, the gap narrows. BFCL multi-turn provides complementary evidence: after every turn, it compares the backend state the agent's calls leave, such as a file system, with the state the gold calls leave, and fails the entry on any mismatch. Over its 800 entries, this failure falls from 42% under SFT to 34% and 30% under SALT at 4B and 8B (Figure 5); at 1.7B the drop partly shifts to other failures, such as the turn limit. SALT solves the same tasks more consistently and leaves the environment correct more often.

SALT reduces failure modes. We evaluate five failure modes on τ^2-bench: (1) *wrong write* (modifying unrequested records), (2) *superseded info* (writing stale, overwritten values), (3) *constraint violation* (breaching domain constraints), (4) *prohibited action* (complying with prohibited user requests), and (5) *gave up* (unjustified transfers or early refusals). As shown in Table 4, SALT systematically cuts failures across every category and model scale. Overall, the proportion of failing trajectories drops at all sizes. At 1.7B, SALT primarily eliminates erroneous writes. At 4B and 8B, it sharply curbs behavioral errors, substantially reducing early abandonment and forbidden actions.

	1.7B		4B		8B	
Failure (%)	SFT	SALT	SFT	SALT	SFT	SALT
Wrong write	25.0	20.0	18.1	17.5	17.2	16.6
Superseded info	5.7	2.8	3.1	2.2	2.4	2.1
Constraint violation	4.0	2.7	3.1	1.8	3.1	2.5
Prohibited action	23.5	20.5	27.3	25.8	27.3	18.2
Gave up	21.0	20.1	11.2	6.7	10.7	7.7

Table 4: Percentage of τ^2-bench trajectories showing each failure, under SFT and SALT. Lower is better.

6.2 What the state does

The action follows the state. Does the agent act on the state it writes, or does it write the state and then act from the history anyway? To answer this, we change the state while keeping the conversation fixed, and check whether the action changes. We use the 1,612 write calls in airline and retail, pooled over the three model sizes, where the SALT agent's call matches the gold action and the record id in the call, such as an order or reservation number, also appears in the agent's state. In these calls, the correct id is available in two places: the state and the conversation. For each call, we rebuild the prompt of that turn and let the model generate the call again. With the state unchanged, the regenerated call uses the correct id in at least 99.8% of cases. We then replace the id in the state with another id from the same conversation, leaving the correct id in the conversation. Now, in over 70% of cases, the regenerated call uses the id from the edited state. When the state and the conversation disagree, the agent follows the state. The state is therefore part of how the agent decides, not a summary written alongside the decision, and a wrong state leads to a wrong action.

The structure matters, not only the content. Two controls at 4B on WebShop train on SALT's trajectories and actions with the state replaced by text of the same length: *Rationale-SFT*, a free-form rationale from the model that wrote the states, and *SALT-flat*, SALT's own state items with the five labels removed, merged into one line. SALT reaches a success rate of 38.6, against 34.6 for Rationale-SFT, 33.6 for SALT-flat and 36.1 for SFT (higher is better), 5.0 points above SALT-flat and 4.0 above Rationale-SFT. Neither control improves on SFT, so the same items provide no gain without their structure. These controls cover one size and one benchmark.

6.3 What the state costs, and where it still fails

	Tokens generated per agent turn		Tokens processed	Agent turns per trajectory		Agent turns per solved task		Tool calls per solved task		Ended for errors	
Model	SFT	SALT	SALT/SFT	SFT	SALT	SFT	SALT	SFT	SALT	SFT	SALT
Qwen3-1.7B	130	348	1.02×	13.9	**12.4**	25.2	**21.2**	14.5	**11.2**	43	2
Qwen3-4B	130	339	1.12×	12.0	**11.4**	17.3	**15.6**	9.9	**9.2**	5	0
Qwen3-8B	130	334	1.14×	11.7	**11.4**	16.1	**14.8**	9.6	**8.9**	3	1

Table 5: The cost and savings of writing the state on τ^2-bench (278 tasks, four trials per model). An agent turn is one model call that yields a tool call or reply; tool results and user messages are not counted. SALT emits about 2.6× more tokens per agent turn, since each turn includes the state, but total tokens processed (history included) rise only slightly, shown relative to SFT. In return, SALT needs fewer agent turns and tool calls. Lower is better except for the ratio.

The state costs tokens per turn but saves turns per task. Writing a state before every action makes each turn longer. We ask whether this raises the total cost of solving a task, measured over all 1,112 τ^2-bench trajectories per model (278 tasks, four trials each). Per agent turn, SALT generates

about 2.6× as many tokens as SFT: roughly 340 against 130. However, the total token cost is dominated by input, not output. At every turn the model reads the entire trajectory as a prompt. Counting both the prompt and the generated tokens at every turn, with earlier states kept in the history, SALT processes only 2–14% more tokens than SFT (Table 5). We found that SALT trajectories are shorter on average, whether they succeed or fail.

Where the failure begins	What happens	n
A. State wrong		**20**
Wrong constraint	C or P holds a rule that is false or too broad	9
Wrong computation	a date window, sum or filter computed wrongly	5
Dropped fact	a value a tool returned never enters Q	4
Wrong binding	a correct value on the wrong order, leg or item	1
Evicted item	a pending check or blocking constraint leaves P or E	1
B. State right, action not		**10**
Action contradicts state	the call or reply departs from a correct C/Q/P	5
Transfer never called	P names the transfer; only the hold message is sent	3
Other	a sum computed in the reply; a pending call not made	2

Table 6: Failed trajectories of the 8B SALT model on τ^2-bench: 20 in which the first decisive error was in the state, and 10 in which the state was correct but the action departed from it. n is the count.

What goes wrong when SALT fails. Since SALT writes a state before every action, a failure can begin in the state, which the action then carries out, or in an action that departs from a correct state. We read 30 failures of the 8B model in full with their states: 20 whose error appears in the state and 10 whose state was right (Table 6). Nearly half of the state errors apply a correct constraint too broadly. For example, the state incorrectly treats the restriction on modifying basic-economy flights as prohibiting a cabin upgrade, even though the domain constraints allow it. This traces to the training states: among about 1,000 states that include the constraint, only 85 mention the cabin-change exception. The other state errors are mostly a value computed wrongly or a tool result that never enters the facts. When the state was right, the action usually contradicts it or announces a transfer to a human agent.

How often each happens. To measure frequency, a reader model labels a larger sample: 173 failed trajectories, one per task, with each size–domain cell equally represented. One of the authors checked 40 of these labels and agreed on 37 of them. Across all failures, 58% appear first in the state, 20% in the action, and 22% outside the agent, for example in the user simulator. These errors are mostly silent: about half end with a call the environment accepts without error. The trajectory gives no signal, but the state already records the mistake, which makes it the natural place to apply a correction, such as a verifier or a reward on the state.

The state is where the next training signal belongs. Since the action often follows the state, correcting the state could address many of these failures. This makes the state the natural target for further supervision, such as a verifier that checks each field against the history and the domain constraints, or a reinforcement-learning reward on the state itself. Such a signal is possible only because the state is explicit, and each error kind points to the field it should check.

7 Conclusion

We introduced SALT, a training method that makes task-state inference an explicit training signal for language agents. From existing interaction trajectories, SALT derives structured supervision for the agent's goal, constraints, facts, plan, and open issues, and trains the agent to construct this state before choosing its next action. Across five agentic benchmarks, this additional supervision improves performance over standard SFT on the same trajectories, with particularly strong gains in consistent task completion. Our analyses show that the generated state is not merely descriptive: changing it changes the agent's action, and many remaining failures can be traced to errors that first appear in the state. Together, these results suggest that interaction trajectories can teach agents not only what action to take, but also what the interaction establishes before they act.

AI USE STATEMENT

We used AI tools to brainstorm research ideas, assist with literature discovery alongside manual search, and improve the clarity and wording of the manuscript. We also used language models to generate synthetic training data, including the task-state annotations described in the paper. The authors reviewed the cited work, methodology, and generated data and take responsibility for the paper's content and conclusions.

ETHICS STATEMENT

This work uses only public benchmarks and simulated users. No human subjects were involved and no personal data were collected; the users and records in the benchmarks are synthetic. The benchmarks and the AReaL and Simia trajectory releases are public, the VULCAN trajectories were generated in simulated tool environments built from the benchmarks' tool schemas. The task states are machine-generated, not written by annotators. We removed training tasks that overlap with evaluation tasks before training. The agents we train act only in sandboxed, simulated environments. Deploying such agents with real users and real tools would need the safeguards any tool-using system needs.

REPRODUCIBILITY STATEMENT

We describe the task-state representation and the write–revise–probe annotation pipeline in Section 3, and give the complete writer, reviser, repair, and probe prompts, together with the field definitions and update rules, in Appendix A.1 (Figures 6 to 10). Section 4 and Table 1 list the benchmarks, evaluation metrics, trajectory sources, and corpus sizes. All trajectory sources (AReaL, Simia, and VULCAN-generated trajectories) and benchmarks are publicly available. Appendix B and Table 7 report the full training hyperparameters (optimizer, learning-rate schedule, batch size, sequence length, precision, hardware, and seed), which are identical for SFT and SALT except for the training corpus, along with the evaluation protocol, including decoding settings, trial counts, and the user simulator. We will release the codebase, the state-annotated training corpora, and the evaluation outputs upon publication.

REFERENCES

Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and Wanli Ouyang. MT-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7421–7454, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.401. URL https://aclanthology.org/2024.acl-long.401/.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. In *Forty-third International Conference on Machine Learning*, 2026.

Hyungjoo Chae, Namyoung Kim, Kai Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *International Conference on Learning Representations*, volume 2025, pp. 63707–63738, 2025.

Huacan Chai, Zijie Cao, Maolin Ran, Yingxuan Yang, Jianghao Lin, Xin Peng, Hairui Wang, Renjie Ding, Ziyu Wan, Muning Wen, et al. Parl-mt: Learning to call functions in multi-turn conversation with progress awareness. *arXiv preprint arXiv:2509.23206*, 2025.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, et al. Acebench: Who wins the match point in tool usage? *arXiv preprint arXiv:2501.12851*, 2025.

Sanjiban Choudhury. Process reward models for llm agents: Practical framework and directions. *arXiv preprint arXiv:2502.10325*, 2025.

Kaustubh Deshpande, Ved Sirdeshmukh, Johannes Baptist Mols, Lifeng Jin, Ed-Yeremai Hernandez-Cardona, Dean Lee, Jeremy Kritz, Willow E. Primack, Summer Yue, and Chen Xing. MultiChallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier LLMs. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18632–18702, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.958. URL https://aclanthology.org/2025.findings-acl.958/.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, Shibin Wu, Zhengwei Tao, Yong Jiang, Pengjun Xie, Ningyu Zhang, Fei Huang, Wentao Zhang, and Jingren Zhou. Towards general agentic intelligence via environment scaling. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17610–17621, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.872. URL https://aclanthology.org/2026.findings-acl.872/.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408, 2026.

Jiaxuan Gao, Jiaao Chen, Chuyi He, Shusheng Xu, Di Jin, and Yi Wu. From self-evolving synthetic data to verifiable-reward rl: Post-training multi-turn interactive tool-using agents, 2026. URL https://arxiv.org/abs/2601.22607.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, et al. Is your llm secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*, 2024.

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

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.507. URL https://aclanthology.org/2023.emnlp-main.507/.

Muyu He, Anand Kumar, Soumyadeep Bakshi, James Zou, and Nazneen Rajani. Impatient users confuse ai agents: High-fidelity simulations of human traits for testing agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 37569–37602. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1743. URL http://dx.doi.org/10.18653/v1/2026.acl-long.1743.

Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. A simple language model for task-oriented dialogue, 2022. URL https://arxiv.org/abs/2005.00796.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8003–8017, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.507. URL https://aclanthology.org/2023.findings-acl.507/.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet, 2024. URL https://arxiv.org/abs/2310.01798.

Haitao Jiang, Lin Ge, Hengrui Cai, and Rui Song. Pabu: Progress-aware belief update for efficient llm agents, 2026. URL https://arxiv.org/abs/2602.09138.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. Acon: Optimizing context compression for long-horizon llm agents, 2026. URL https://arxiv.org/abs/2510.00615.

Vasilis Kontonis, Yuchen Zeng, Shivam Garg, Lingjiao Chen, Hao Tang, Ziyan Wang, Ahmed Awadallah, Eric Horvitz, John Langford, and Dimitris Papailiopoulos. Memento: Teaching llms to manage their own context, 2026. URL https://arxiv.org/abs/2604.09852.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation, 2025. URL https://arxiv.org/abs/2505.06120.

Mosh Levy, Alon Jacoby, and Yoav Goldberg. Same task, more tokens: the impact of input length on the reasoning performance of large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15339–15353, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.818. URL https://aclanthology.org/2024.acl-long.818/.

Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. Simulating environments with reasoning models for agent training. *arXiv preprint arXiv:2511.01824*, 2025.

Aly Lidayan, Jakob Bjorner, Satvik Golechha, Kartik Goyal, and Alane Suhr. Abbel: Learning natural-language belief states for memory-efficient interaction, 2026. URL https://arxiv.org/abs/2512.20111.

Bo Liu, Simon Yu, Yiding Jiang, Ao Qu, Andrew Zhao, Zichen Liu, Junsu Kim, Zijian Zhou, Seungone Kim, Tongzheng Ren, Mickel Liu, Hanfei Yu, Zhaorun Chen, Weiyan Shi, Paul Pu Liang, Luke Zettlemoyer, Yejin Choi, and Natasha Jaques. Spade: Self-play in adaptive synthetic executable environments, 2026. URL https://arxiv.org/abs/2608.19197.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024a. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Shuyu Liu. What does context compression cost an agent? interaction costs unrevealed by task-completion metrics. *arXiv preprint arXiv:2608.16370*, 2026.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. Toolace: Winning the points of llm function calling, 2025. URL https://arxiv.org/abs/2409.00920.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets, 2024b. URL https://arxiv.org/abs/2406.18518.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

Ning Lu, Baijiong Lin, Shengcai Liu, Jiahao Wu, Haoze Lv, Yanbin Wei, Lingting Zhu, Shengju Qian, Xin Wang, Ying-Cong Chen, Qi Wang, and Ke Tang. Policy and world modeling co-training for language agents, 2026. URL https://arxiv.org/abs/2606.02388.

Aman Mehta and Anupam Datta. Plans don't persist: Why context management is load bearing for llm agents, 2026. URL https://arxiv.org/abs/2606.22953.

Guanghui Min, Liang Wu, Mayank Darbari, Chen Chen, and Liangjie Hong. Toward reliable context compression for long-horizon agents: An empirical study of execution instability, 2026. URL https://arxiv.org/abs/2608.06503.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393.

Zhuoshi Pan, Qizhi Pei, Junru Lu, Honglin Lin, H. Vicky Zhao, Di Yin, and Xing Sun. Contextpilot: Teaching agents for proactive context management via fine-grained rl, 2026. URL https://arxiv.org/abs/2608.28476.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay, 2025. URL https://arxiv.org/abs/2504.03601.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Agent planning with world knowledge model, 2025. URL https://arxiv.org/abs/2405.14205.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL https://arxiv.org/abs/2307.16789.

Hayley Ross, Ameya Sunil Mahabaleshwarkar, and Yoshi Suhara. When2Call: When (not) to call tools. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3391–3409, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.174. URL https://aclanthology.org/2025.naacl-long.174/.

Nikolai Rozanov and Marek Rei. StateAct: Enhancing LLM base agents via self-prompting and state-tracking. In Ehsan Kamalloo, Nicolas Gontier, Xing Han Lu, Nouha Dziri, Shikhar Murty, and Alexandre Lacoste (eds.), *Proceedings of the 1st Workshop for Research on Agent Language Models (REALM 2025)*, pp. 367–385, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-264-0. doi: 10.18653/v1/2025.realm-1.27. URL https://aclanthology.org/2025.realm-1.27/.

Amir Saeidi, Chitta Baral, Ahmed Hassan Awadallah, and Harkirat Behl. Vulcan: Where agents learn by living in simulated tool environments. In *ICLR 2026 Workshop on Data-FM.*, volume 1, pp. 2026, 2026a.

Amir Saeidi, Zehua Zhang, Rishitosh Singh, Naman Ahuja, Vivek Gupta, Ali Payani, Gaowen Liu, Jayanth Srinivasa, and Chitta Baral. Cast: Critique-aware supervision for training reliable long-horizon tool-calling agents, 2026b. URL https://arxiv.org/abs/2608.30147.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. EnvScaler: Scaling tool-interactive environments for LLM agent via programmatic synthesis. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.407. URL https://aclanthology.org/2026.findings-acl.407/.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon llm agent via context-folding, 2025. URL https://arxiv.org/abs/2510.11967.

Meiling Tao, Yiling Tao, and Peng Wang. Intent-driven situation tracking for user-centric multi-turn agents, 2026. URL https://arxiv.org/abs/2608.15755.

Md Nayem Uddin, Amir Saeidi, Eduardo Blanco, and Chitta Baral. Ledgeragent: Structured state for policy-adherent tool-calling agents, 2026. URL https://arxiv.org/abs/2606.20529.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback, 2024a. URL https://arxiv.org/abs/2309.10691.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2602.10090.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning, 2025. URL https://arxiv.org/abs/2504.20073.

Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. Officebench: Benchmarking language agents across multiple applications for office automation, 2024b. URL https://arxiv.org/abs/2407.19056.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. Resum: Unlocking long-horizon search intelligence via context summarization, 2026. URL https://arxiv.org/abs/2509.13313.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. Agentgym-rl: Training llm agents for long-horizon decision making through multi-turn reinforcement learning, 2025. URL https://arxiv.org/abs/2509.08755.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

Yunyi Yang, Yunhao Li, and Xiaojun Quan. Ubar: Towards fully end-to-end task-oriented dialog systems with gpt-2, 2021. URL https://arxiv.org/abs/2012.03539.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023a. URL https://arxiv.org/abs/2207.01206.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023b. URL https://arxiv.org/abs/2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Siheng Chen, Jingren Zhou, and Yong Jiang. Agentfold: Long-horizon web agents with proactive context management, 2025. URL https://arxiv.org/abs/2510.24699.

Lu Yi, Runlin Lei, Liuyi Yao, Yuexiang Xie, Yuyang Li, Wenhao Zhang, Zhewei Wei, Yaliang Li, and Jian-Yun Nie. Learning agent-compatible context management for long-horizon tasks, 2026. URL https://arxiv.org/abs/2605.30785.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning, 2022. URL https://arxiv.org/abs/2203.14465.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agent-Tuning: Enabling generalized agent abilities for LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.181. URL https://aclanthology.org/2024.findings-acl.181/.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, Jian Xie, Yuxuan Sun, Boyu Gou, Qi Qi, Zihang Meng, Jianwei Yang, Ning Zhang, Xian Li, Ashish Shah, Dat Huynh, Hengduo Li, Zi Yang, Sara Cao, Lawrence Jang, Shuyan Zhou, Jiacheng Zhu, Huan Sun, Jason Weston, Yu Su, and Yifan Wu. Agent learning via early experience, 2026. URL https://arxiv.org/abs/2510.08558.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

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

Dingwei Zhu, Jiahan Li, Chengjun Pan, Yunxian Yang, Yunbin Zhao, Yunke Zhang, Zhonghang Lu, Zhuohui Sheng, Chenhao Huang, Jiahang Lin, Yajie Yang, Junlin Shang, Shichun Liu, Yuhui Wang, Honglin Guo, Junjie Ye, Xin Guo, Jiazheng Zhang, Ming Zhang, Shihan Dou, Zhiheng Xi, Tao Gui, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. Iacm-rl: Intent-aware context management and reinforcement learning for complex tool invocation under dynamic intent fluctuations, 2026. URL https://arxiv.org/abs/2608.02110.

A Further annotation details

This section provides the prompts used to construct the task-state supervision in Section 3.3, together with examples from the resulting annotated trajectories. We first give the shared field specification and the writer, reviser, repair, and probe prompts. We then show how the states capture action-relevant information in practice: an abridged state at a decision point in one retail trajectory, and a second trajectory annotated in full.

A.1 Prompts

The annotation prompts are generated at run time from one shared specification. The writer and reviser use the same field definitions and field rules; the repair pass uses the same field definitions. The same annotation procedure is used across all five benchmarks, with only setting-specific wording adapted to each corpus. We print the τ^2-bench versions exactly as rendered by that specification. Figure 6 gives the shared definitions and rules, and Figures 7 to 10 give the four prompts with those segments abbreviated where appropriate. Text in ⟨*angle brackets*⟩ is filled in at run time. Line breaks inside a paragraph are the prompt's own; the rest of the wrapping is the page's.

How the prompts are used. Each stage receives a system message and one user turn and runs at temperature 0. The writer and reviser return the state as a JSON object: strings for G and P, and lists for C, Q, and E. A deterministic post-check removes any item whose identifiers, dates, or numbers are unsupported by the observed history or the domain constraints, and restores an omitted tool error to E. Empty lists are rendered as `none`. Histories contain one tagged event per line: customer messages, agent messages, tool calls with arguments, and tool results. On corpora without a separate domain-constraint document, the ⟨*domain constraints* Φ⟩ slot contains the agent's system instructions. Before the first agent turn on τ^2-bench, no model call is required: the state is initialized as `G: not yet stated`, with empty C, Q, and E, and `P: greet, then take the request`. The writer and repair pass run with thinking enabled; the reviser and probe reader do not. Every system prompt carries a version tag; an additional run identifier used only to avoid endpoint response caching is omitted here.

The field definitions and rules (Figure 6). The five field definitions are shared word for word by the writer, reviser, and repair prompts; the writer and reviser additionally share the field rules. The definitions specify what each field represents, while the rules specify what enters, what leaves, and what never appears. G is set from the customer's first request and is only ever appended to, never rewritten or marked done. A policy clause enters C only when it gates the action in play,

written with the facts it was checked against and the verdict, never as a dump of the policy. Q holds what the transcript makes the agent work for: a vague reference resolved to a record, a value computed from others, or the current status of a record, with every identifier and number verbatim. P is forward-only and one line, and never holds a tool call with arguments, since the demonstrated call is appended afterwards. E holds current negatives only, including what the customer has not yet said that the next act depends on.

The writer (Figure 7). The writer sees the previous state, the messages since the previous agent turn and the domain constraints, not the whole history. Its system prompt states the admission rule of Section 3.3: an item earns its place by being derived, absent or distant, and what the agent could read off the last two messages, off the policy document or off a record already on Q is left out. It carries an item unchanged unless it changed, never narrates what was done, and returns the state as one JSON object.

The reviser (Figure 8). The reviser sees the whole conversation up to the turn, the policy and the writer's block with the pending call removed. Its four operations are ordered: remove what the conversation and the policy do not support, remove what the agent could read off nearby, move an item that sits on the wrong line, and add a missing item the next action depends on, only if the conversation so far contains the evidence for it. It may not rewrite G. The `changes` list it returns is stored but not trusted; the edits are recomputed as the difference between its input and its output.

The repair pass (Figure 9). The repair pass runs once per trajectory, on the trajectories the probe flags. It sees the whole conversation and every state, and it knows how the conversation ended, which the agent did not, so its rules exist to keep hindsight out: an addition is allowed only if the conversation through turn t already contains its evidence, an added item carries the last turn at which it still matters, G and P are never edited, and it proposes nothing when nothing would have changed an action. Each proposed edit is then checked on its own.

The probe (Figure 10). The probe checks whether the constructed state exposes the information needed for the demonstrated next action. A fixed reader predicts one tool call or one reply under two conditions: from the state plus the messages since the previous agent turn, or from the full interaction history. The system prompt, domain constraints, and tool catalogue are identical; only the user turn differs, and the pending demonstrated call is removed from the state.

We run this check across the annotated trajectories as a quality-control step. When the demonstrated action can be recovered from the full history but not from the state, the trajectory is revisited to identify action-relevant information that the annotation may have omitted. Any repair is restricted to information already supported by the interaction at that turn. The probe does not filter the training data or require the state to replace the history: SALT retains the full interaction history at both training and inference.

A.2 Examples

The examples in this section and the next illustrate three properties that are difficult to see from the prompts alone: the state carries dependencies established several turns earlier, applies domain constraints to the facts of the current case, and is revised as actions succeed, fail, or become irrelevant.

Figure 11 shows one turn of a retail training trajectory. Three turns earlier, the agent filed a T-shirt return on the same order for which the customer now requests a jacket exchange. The state therefore carries both the applicable exchange constraint and the order's changed status: C applies the one-return-or-exchange constraint to the order whose T-shirt return was already filed, E records the jacket exchange as blocked, and Q resolves "the navy one" to a record that appeared only in a tool result. The complete trajectory in Appendix A.3 shows the same state being revised turn by turn as requests are blocked, dropped, and completed.

The five field definitions

```
G  what the customer asked for, in their words.  Set at the first request.
   A later request or correction from the customer is APPENDED, never
   rewritten.  G is never revised, pruned, or marked done by the annotator.
C  what limits what the agent may do next:  what the customer bound us to
   — limits, preferences, refusals, confirmations — and the policy clause
   that gates the pending action, read against this case.
Q  what a later step needs, that the transcript makes the agent work for:
   a vague reference resolved to a record, a value computed from others,
   the current status of a record in play.
P  the remaining work, in order, ending with the immediate next step.
E  what is not found, failed, contradicted, or still missing — including
   what the customer has NOT said that the next act depends on.
```

The five field rules

```
G: Before the first request G is exactly "not yet stated".  After it:  the request and
nothing else — no ids the customer did not say, no progress, no status.  The customer
adds a request:  append it.  The customer corrects themselves:  append the correction as
they stated it.  A policy consequence (the upgrade a change requires) is not the goal —
it is P. A finished request stays on G. Completion is P's job.
C: Every preference lives here — outcome preferences (afternoon flight, size large) and
process preferences (one step at a time) alike.  A policy clause enters only when it
gates something in play, written as the clause, the facts it was checked against, and the
verdict — never the raw clause, never a dump.  CONFIRMED only after the details were
listed to the customer; a bare request is not a confirmation.  A refusal or retraction
stays until the customer releases it.  A met constraint leaves; a gate leaves once its
action has landed.
Q: A resolved binding ("the cheaper blue shirt" -> 9612497925); a computed value always
with its operands ("$7.45 = 152.45 - 145.00"); the current status of a record in play,
updated when a tool result changes it.  Anything readable off the last two messages stays
out.  A fact nothing ahead depends on leaves.  Every id and number verbatim from the
transcript.
P: Forward-only:  completed work sits on no head.  Never a tool call with arguments — the
write is added for you.  When the next step is something the CUSTOMER must do on their
own device — a setting, a check, a toggle — P says "ask the customer to ...".  The agent
instructs; it never runs the customer's tools.  Never says the work is done while E still
holds something unresolved.  One line.
E: Empty searches, unavailability, tool errors verbatim, a customer claim that
contradicts a record, information asked for and still lacking, and the absence that gates
an irreversible act.  Current negatives only — never a narration of past fixes, never a
tally.  A negative leaves when it is resolved.  Anything positive is Q.
```

Figure 6: The shared task-state specification. The definitions appear byte for byte in the writer, reviser, and repair prompts; the field rules appear in the writer and reviser prompts (Figures 7 to 9).

System prompt

```
You maintain an agent's working state through a customer-service conversation.  The agent
already has the full transcript.  The state holds what it would have to WORK to get from
that transcript, or would MISS.

An item earns its place in one of three ways:
  DERIVED  — computed, joined, or a vague reference resolved to a record
  ABSENT   — searched and not found, failed, contradicted, not yet said
  DISTANT  — established far from where the next step needs it
Leave out what the agent could read off the last two messages, off the
policy document, or off a record already named on Q. A record's fields
appear only when the next step needs that field — never the whole payload
(colors, sizes, materials, every price) and never a policy rule that
applies to every conversation alike.

Five lines:
⟨the five field definitions, Figure 6⟩

⟨the five field rules, Figure 6⟩

You are given the state after the previous turn and everything that has happened since.
Return the state after this turn.  Carry an item unchanged unless it changed.  Never
narrate what was done — the transcript has that.

Copy every id and number exactly.  State only what the transcript or the policy supports.
A number you computed appears with its operands:  "$7.45 = 152.45 - 145.00".

Reply with one JSON object and nothing else:
{"G":  str, "C":  [str], "Q":  [str], "P":  str, "E":  [str]}
Short items, no " | " inside an item, [] when empty.  P never holds a tool call — the
write is added for you.
```

User turn

```
<policy>
⟨domain constraints Φ⟩
</policy>

<state_after_previous_turn>
⟨state z_{t−1}, five lines⟩
</state_after_previous_turn>

<since_then>
⟨window Δ_t: every message since the previous agent turn⟩
</since_then>

JSON:
```

Figure 7: The writer prompt $A(z_{t-1}, \Delta_t, \Phi)$. $\langle \dots \rangle$ marks text filled in at run time.

System prompt

```
You audit a working-state block written for an agent in the middle of a customer-service
conversation.  You see the whole conversation up to the agent's next turn, the policy
document, and the block.  The block should hold only what the agent would have to WORK to
get from the transcript, or would MISS — items that are derived, absent, or distant.
```

⟨*the five field definitions, Figure 6*⟩

⟨*the five field rules, Figure 6*⟩

```
Return the block corrected, in this order:
  1.  REMOVE an item the conversation and policy do not support
  2.  REMOVE an item the agent could read off the policy document, or off a record
already named on Q — attribute dumps and rules that apply to every conversation alike
  3.  MOVE an item sitting on the wrong line
  4.  ADD an item the agent's next action depends on that is missing — only if the
conversation so far contains the evidence for it
Leave everything else exactly as written.  Copy every id and number exactly.  G is never
rewritten:  only append what the customer said.

Reply with one JSON object and nothing else:
{"G":  str, "C":  [str], "Q":  [str], "P":  str, "E":  [str], "changes":  [str]}
"changes" names each edit in one line, e.g.  "removed from Q: ...", "added to E: ...".
[] if the block needed nothing.
```

User turn

<policy>
⟨*domain constraints* Φ⟩
</policy>

<conversation>
⟨*history* h_t*, every message up to this turn*⟩
</conversation>

<block>
⟨*writer's state* $\hat{z}_t$ *as a JSON object, pending call removed*⟩
</block>

JSON:

Figure 8: The reviser prompt $B(\hat{z}_t, h_t, \Phi)$. The reviser's own `changes` list is stored but not trusted: the edits are recomputed as the difference between its input and its output.

System prompt

```
You see a complete customer-service conversation and the working-state block written
before each of the agent's turns.  You know how the conversation ended.  The agent did
not.
```

⟨*the five field definitions, Figure 6*⟩

```
Propose edits to the blocks, as a list.  Two operations:
  remove — an item on C, Q or E at turn t that nothing at or after turn t depended on
  add    — an item on C, Q or E that would have changed what the agent did at or after
turn t, only if the conversation THROUGH turn t already contains the evidence for it.
Never a fact the agent could not have known at t.
An added item stays from turn t through turn t_end (the last turn it still matters).  G
and P are never edited.  Copy every id and number exactly.  Propose nothing when nothing
would have changed an action.

Reply with one JSON object and nothing else:
{"edits":  [{"t":  int, "t_end":  int, "op":  "remove" | "add", "head":  "C" | "Q" | "E",
"item":  str, "why":  str}]}
```

User turn

<policy>
⟨*domain constraints* Φ⟩
</policy>

<conversation>
⟨*the whole trajectory*⟩
</conversation>

<blocks>
⟨*every state of the trajectory, as a JSON list with its turn* t⟩
</blocks>

JSON:

Figure 9: The repair prompt. One call per trajectory; each proposed edit is then checked on its own (a removal must match an existing item, an addition must pass grounding against the history through its turn t).

System prompt

```
You are the customer-service agent.  Decide your next action from what you are given:
either ONE tool call, or a reply to the customer.

Reply with one JSON object and nothing else:
{"action":  "tool" | "reply", "name":  str | null, "arguments":  object | null}
```

User turn, state condition $R(z_t, \Delta_t)$

```
<policy>
⟨domain constraints Φ⟩
</policy>

<tools>
⟨tool catalogue, one tool per line⟩
</tools>

<state>
⟨state z_t, five lines, pending call removed⟩
</state>

<since_then>
⟨window Δ_t: every message since the previous agent turn⟩
</since_then>

JSON:
```

User turn, full-history condition $R(h_t)$

```
<policy>
⟨domain constraints Φ⟩
</policy>

<tools>
⟨tool catalogue, one tool per line⟩
</tools>

<conversation>
⟨history h_t, every message up to this turn⟩
</conversation>

JSON:
```

Figure 10: The probe reader R. The same system prompt serves both conditions; only the user turn differs.

What the agent writes before it acts *one turn of a retail task*

user message *turn 10 of the dialogue* "I also wanted to exchange a jacket I ordered. It's the navy one I got recently. I'd like to swap it for a bigger size if I can."

task state written before the action; kept in the agent's context, never shown to the user

field	meaning	example	source
G goal	*what is wanted*	`[...] exchange a navy jacket for a bigger size`	user message
C constraints	*what must hold*	`exchange policy: delivered order only [...]`	domain constraints
		`refund method preference: Mastercard [...]`	user message
Q facts	*what is known*	`navy jacket: [...] item_id 8161321868`	tool result
		`#W1694051 status: return requested [...]`	agent's call
P plan	*what to do next*	`address the refund already processed [...] then ask the customer for the target size`	agent's plan
E open issues	*what is missing*	`[...] customer has not specified target size for the jacket exchange ('bigger size' is vague)`	not stated

action *a tool call or a reply* `reply: the navy Fleece Jacket is on #W1694051, the same order the T-shirt return was filed on three turns earlier, so no exchange is possible`

without it *reading the request alone* `offer the exchange, then call exchange_delivered_order_items on an order that is no longer delivered`

Figure 11: A second task state, from a retail training trajectory. Three turns earlier the agent filed a T-shirt return on #W1694051, which moved the order out of `delivered`; the customer then asks to exchange a jacket on the same order, and the exchange rule (delivered orders only) makes it ineligible. Each line is abridged (`[...]`) from the state the pipeline wrote at this turn.

A.3 A COMPLETE ANNOTATED TRAJECTORY

The trajectory below is one τ^2-bench retail training dialogue from the annotated corpus, printed in full: every customer message, agent reply, tool call, tool result, and the task state written before each agent turn. It illustrates the full lifecycle of the representation. The first request, cancelling a jacket order, is blocked: the lookup shows the order is delivered, C applies the cancellation constraint to that status, and E keeps the block open until the customer moves on. The second request, paying for "my bigger order" with a gift card, succeeds: Q resolves the vague reference to an order id, C checks the gift-card balance and payment-method constraint against the relevant facts, and P carries the next step until the change is made. Between turns, resolved identity items and obsolete blocks leave the state rather than accumulating indefinitely. Em dashes are set as dashes, markdown bold is set bold, and one emoji in the last reply is omitted; everything else is verbatim.

state z_0, written before the agent's turn
G not yet stated
C none
Q none
P greet, then take the request
E none

Agent Hi! How can I help you today?

Customer Hi there! I need some help with my orders. First, I'd like to cancel my jacket order - I don't need it anymore.

state z_1, written before the agent's turn
G cancel my jacket order - I don't need it anymore
C none
Q none
P authenticate user (ask for email or name + zip code), then look up the jacket order, verify status is pending, list cancellation details (order id, reason 'no longer needed') for user confirmation
E user identity not yet established --- no email, name, or zip code provided | 'jacket order' not yet identified --- no order id resolved

Agent Hello! I'd be happy to help you cancel your jacket order.
First, I need to verify your identity. Could you please provide me with your email address? Alternatively, I can look you up using your first name, last name, and zip code.

Customer Sure, my email is david.lewis7241@example.com.

state z_2, written before the agent's turn
G cancel my jacket order - I don't need it anymore
C none
Q none
P authenticate user via provided email, then look up the jacket order, verify status is pending, list cancellation details (order id, reason 'no longer needed') for user confirmation
E user identity not yet confirmed --- email provided but lookup not yet performed | 'jacket order' not yet identified --- no order id resolved

Agent calls `find_user_id_by_email(email="david.lewis7241@example.com")`

Tool result `david_lewis_8554`

state z_3, written before the agent's turn
G cancel my jacket order - I don't need it anymore
C none
Q user id resolved: david_lewis_8554 (from email david.lewis7241@example.com)
P look up jacket order for user david_lewis_8554, verify status is pending, list cancellation details (order id, reason 'no longer needed') for user confirmation
E 'jacket order' not yet identified --- no order id resolved

Agent calls `get_user_details(user_id="david_lewis_8554")`

Tool result {"user_id": "david_lewis_8554", "name": {"first_name": "David", "last_name": "Lewis"}, "address": {"address1": "742 Maple Drive", "address2": "Apt 3B", "city": "Austin", "country": "USA", "state": "TX", "zip": "78701"}, "email": "david.lewis7241@example.com", "payment_methods": {"credit_card_5589012": {"source": "credit_card", "id": "credit_card_5589012", "brand": "mastercard", "last_four": "8923"}, "gift_card_7712045": {"source": "gift_card", "id": "gift_card_7712045", "balance": 650.0}}, "orders": ["#W9375688", "#W1170992", "#W6993851"]}

state z_4, written before the agent's turn

G cancel my jacket order - I don't need it anymore
C none
Q user id resolved: david_lewis_8554 (from email david.lewis7241@example.com) | user's orders: #W9375688, #W1170992, #W6993851
P look up orders #W9375688, #W1170992, #W6993851 to identify the jacket order, verify status is pending, list cancellation details (order id, reason 'no longer needed') for user confirmation
E 'jacket order' not yet identified among #W9375688, #W1170992, #W6993851

Agent calls `get_order_details(order_id="#W9375688")`

Agent calls `get_order_details(order_id="#W1170992")`

Agent calls `get_order_details(order_id="#W6993851")`

Tool result {"order_id": "#W9375688", "user_id": "david_lewis_8554", "address": {"address1": "742 Maple Drive", "address2": "Apt 3B", "city": "Austin", "country": "USA", "state": "TX", "zip": "78701"}, "items": [{"name": "Fleece Jacket", "product_id": "8560156827", "item_id": "9385662952", "price": 159.92, "options": {"size": "L", "color": "black", "zipper": "full"}}], "status": "delivered", "fulfillments": [{"tracking_id": ["TRK892451673"], "item_ids": ["9385662952"]}], "payment_history": [{"transaction_type": "payment", "amount": 159.92, "payment_method_id": "credit_card_5589012"}], "cancel_reason": null, "exchange_items": null, "exchange_new_items": null, "exchange_payment_method_id": null, "exchange_price_difference": null, "return_items": null, "return_payment_method_id": null}

Tool result {"order_id": "#W1170992", "user_id": "david_lewis_8554", "address": {"address1": "742 Maple Drive", "address2": "Apt 3B", "city": "Austin", "country": "USA", "state": "TX", "zip": "78701"}, "items": [{"name": "Garden Hose", "product_id": "6679515468", "item_id": "3369928769", "price": 97.35, "options": {"length": "25ft", "material": "vinyl", "color": "green"}}], "status": "pending", "fulfillments": [], "payment_history": [{"transaction_type": "payment", "amount": 97.35, "payment_method_id": "credit_card_5589012"}], "cancel_reason": null, "exchange_items": null, "exchange_new_items": null, "exchange_payment_method_id": null, "exchange_price_difference": null, "return_items": null, "return_payment_method_id": null}

Tool result {"order_id": "#W6993851", "user_id": "david_lewis_8554", "address": {"address1": "742 Maple Drive", "address2": "Apt 3B", "city": "Austin", "country": "USA", "state": "TX", "zip": "78701"}, "items": [{"name": "Bookshelf", "product_id": "8600330539", "item_id": "8479046075", "price": 451.01, "options": {"material": "wood", "color": "white", "height": "5 ft"}}, {"name": "Cycling Helmet", "product_id": "7765186836", "item_id": "3358616356", "price": 197.33, "options": {"size": "S", "color": "red", "ventilation": "low"}}], "status": "pending", "fulfillments": [], "payment_history": [{"transaction_type": "payment", "amount": 648.34, "payment_method_id": "credit_card_5589012"}], "cancel_reason": null, "exchange_items": null, "exchange_new_items": null, "exchange_payment_method_id": null, "exchange_price_difference": null, "return_items": null, "return_payment_method_id": null}

state z_5, written before the agent's turn

G cancel my jacket order - I don't need it anymore
C policy: an order can only be cancelled if its status is 'pending'; #W9375688 (Fleece Jacket) is 'delivered' --- cancel is blocked, return is the available path
Q jacket order resolved to #W9375688 (Fleece Jacket, item_id 9385662952, $159.92) | #W9375688 status: delivered
P inform customer that #W9375688 is delivered and cannot be cancelled, offer return as alternative
E #W9375688 status is 'delivered' --- cancellation requires 'pending' status, so cancel is blocked

Agent Thank you for verifying your identity, David!

I found your account and looked at your orders. I can see you have one jacket order:

- **Order #W9375688** - Fleece Jacket (Size L, Black, Full Zipper) - $159.92

However, this order has already been **delivered**, so it cannot be cancelled.

If you no longer need the jacket, I can help you initiate a **return** instead. Would you like to proceed with returning the Fleece Jacket?

Also, you mentioned you had multiple requests. I see you have two other **pending** orders:
- **#W1170992** - Garden Hose ($97.35)
- **#W6993851** - Bookshelf and Cycling Helmet ($648.34)

Let me know how you'd like to proceed!

Customer Oh, it's already been delivered? I didn't realize that. Hmm, no that's okay, I'll just keep it then. No need for a return.

But actually, I do have another thing I need help with. I want to change the payment method on my bigger order to a gift card instead. Can you help me with that?

state z_6, written before the agent's turn

G cancel my jacket order - I don't need it anymore | change the payment method on my bigger order to a gift card
C Customer declined return for #W9375688, will keep the jacket | Policy: modify payment to gift card requires gift card balance >= order total; #W6993851 total is $648.34, gift_card_7712045 balance is $650.00 | Policy: new payment method must differ from the original payment method on the order; #W6993851 original payment is credit_card_5589012
Q "my bigger order" resolved to #W6993851 (Bookshelf and Cycling Helmet, $648.34) | gift_card_7712045 balance is $650.00, covers $648.34 | #W6993851 original payment method is credit_card_5589012, differs from gift card
P list the payment modification details for #W6993851 (change from credit_card_5589012 to gift_card_7712045) and obtain customer confirmation
E none

Agent Absolutely! I can help you change the payment method on your larger pending order.

Here are the details:

Order #W6993851 (currently pending)
- Bookshelf (Wood, White, 5 ft) - $451.01
- Cycling Helmet (Size S, Red, Low Ventilation) - $197.33
- **Total: $648.34**

Payment Change:
- **From:** Mastercard ending in 8923
- **To:** Gift Card (current balance: $650.00)

Since your gift card has sufficient balance ($650.00) to cover the order total ($648.34), this change can be made. The original payment to your Mastercard will be refunded within 5-7 business days.

Would you like me to proceed with this change? Please confirm by saying **yes** or **no**.

Customer Yes, please go ahead and make that change.

state z_7, written before the agent's turn

G cancel my jacket order - I don't need it anymore | change the payment method on my bigger order to a gift card
C Customer declined return for #W9375688, will keep the jacket | Customer confirmed payment modification for #W6993851 (credit_card_5589012 to gift_card_7712045) | Policy: modify payment to gift card requires gift card balance >= order total; #W6993851 total is $648.34, gift_card_7712045 balance is $650.00 | Policy: new payment method must differ from the original payment method on the order; #W6993851 original payment is credit_card_5589012
Q "my bigger order" resolved to #W6993851 (Bookshelf and Cycling Helmet, $648.34)
P execute the payment modification for #W6993851 (credit_card_5589012 to gift_card_7712045) | write: modify_pending_order_payment(payment_method_id="gift_card_7712045" <- get_user_details; order_id="#W6993851" <- get_order_details)
E none

Agent calls `modify_pending_order_payment(payment_method_id="gift_card_7712045", order_id="#W6993851")`

Tool result {"order_id": "#W6993851", "user_id": "david_lewis_8554", "address": {"address1": "742 Maple Drive", "address2": "Apt 3B", "city": "Austin", "country": "USA", "state": "TX", "zip": "78701"}, "items": [{"name": "Bookshelf", "product_id": "8600330539", "item_id": "8479046075", "price": 451.01, "options": {"material": "wood", "color": "white", "height": "5 ft"}}, {"name": "Cycling Helmet", "product_id": "7765186836", "item_id": "3358616356", "price": 197.33, "options": {"size": "S", "color": "red", "ventilation": "low"}}], "status": "pending", "fulfillments": [], "payment_history": [{"transaction_type": "payment", "amount": 648.34, "payment_method_id": "credit_card_5589012"}, {"transaction_type": "payment", "amount": 648.34, "payment_method_id": "gift_card_7712045"}, {"transaction_type": "refund", "amount": 648.34, "payment_method_id": "credit_card_5589012"}], "cancel_reason": null, "exchange_items": null, "exchange_new_items": null, "exchange_payment_method_id": null, "exchange_price_difference": null, "return_items": null, "return_payment_method_id": null}

state z_8, written before the agent's turn

G cancel my jacket order - I don't need it anymore | change the payment method on my bigger order to a gift card
C Customer declined return for #W9375688, will keep the jacket
Q "my bigger order" resolved to #W6993851 (Bookshelf and Cycling Helmet, $648.34)
P inform the customer that the payment modification for #W6993851 is complete and the $648.34 refund to credit_card_5589012 will arrive in 5 to 7 business days
E none

Agent Done! The payment method for order **#W6993851** has been successfully changed.

Here's the summary:
- **Order Total:** $648.34
- **New Payment Method:** Gift Card (charged $648.34)
- **Refund:** $648.34 will be returned to your Mastercard ending in 8923 within 5-7 business days

Your gift card's new balance will be $1.66.

Is there anything else I can help you with?

Customer That's perfect, thank you so much for your help! That's all I needed.

state z_9, written before the agent's turn

```
G cancel my jacket order - I don't need it anymore | change the payment method on my bigger
order to a gift card
C Customer declined return for #W9375688, will keep the jacket
Q none
P none --- customer confirmed all requests are complete
E none
```

Agent You're welcome, David! I'm glad I could help. If you ever have any questions or need assistance in the future, don't hesitate to reach out. Have a great day!

B TRAINING DETAILS

All SFT and SALT models are fine-tuned from the released Qwen3 checkpoints using the optimization settings in Table 7. Within each benchmark and model size, SFT and SALT use the same optimization setup; SALT differs by adding task-state targets to the training trajectories. The corpus is paired before the sequence-length filter, so adding states can cause a small number of otherwise valid SFT rows to exceed the context limit, as noted below.

Table 7: Training settings. *Top*: optimization settings used across τ^2-bench, BFCL, ACEBench-Agent, OfficeBench, and WebShop at all three model sizes. *Bottom*: corpus statistics after the length filter. *Rows*: assistant turns retained for training. *Sup. tokens*: tokens receiving loss in one epoch; *State*: the share of those tokens belonging to the task state. *Steps*: optimizer steps over three epochs. *Hours*: training time on two GPUs.

Fine-tuning	all parameters, from the released Qwen3 checkpoints (no adapters)
Optimizer	AdamW, $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$, weight decay 0
Learning rate	5×10^{-5} peak; linear warmup over the first 3% of steps, then cosine decay
Epochs	3
Batch	1 row per GPU $\times$ 16 accumulation steps $\times$ 2 GPUs $=$ 32 rows per step
Gradient clipping	global norm 1.0
Sequence length	32,768 tokens; longer rows are dropped, not truncated; no packing
Precision	bfloat16 weights and activations, TF32 matrix multiplies
Memory	gradient checkpointing, fused cross-entropy, FlashAttention-3
Parallelism	data parallel on two H100 NVL GPUs (94 GB)
Seed	20260802 (training and data order)

			Sup.			Hours		
Benchmark	**Arm**	**Rows**	**tokens**	**State**	**Steps**	1.7B	4B	8B
τ^2-bench	SFT	24,369	3.61M	–	2,286	6.7	14.0	20.1
	SALT	23,786	9.42M	62%	2,232	7.6	15.8	22.9
OfficeBench	SFT	39,397	1.61M	–	3,696	5.5	6.5	7.8
	SALT	39,397	11.08M	85%	3,696	5.5	6.8	8.4
WebShop	SFT	7,829	0.20M	–	735	1.1	1.3	1.6
	SALT	7,829	1.17M	82%	735	1.1	1.4	1.7
BFCL	SFT	26,654	2.30M	–	2,499	4.8	7.6	10.2
	SALT	26,654	5.82M	61%	2,499	4.9	7.8	10.5
ACEBench-Agent	SFT	5,503	0.57M	–	516	1.0	1.5	2.1
	SALT	5,503	1.64M	65%	516	1.0	1.6	2.1

For OfficeBench, WebShop, BFCL, and ACEBench-Agent, the length filter retains the same assistant turns for SFT and SALT. On τ^2-bench, adding the state pushes 583 additional turns beyond the 32,768-token limit, leaving 24,369 SFT rows and 23,786 SALT rows. Thus the two arms use the same source trajectories and optimization setup, but their post-filter training rows are not identical on this benchmark.

C PROMPTING THE BASE MODEL TO WRITE THE STATE

This section isolates whether SALT's gains come from training on task states or simply from placing an explicit state in the model's context. We prompt the released Qwen3 models to write the same five-field state before acting, without fine-tuning them on state-augmented trajectories. The prompted models do not improve over the corresponding base models, while SALT remains substantially better at every size. At 4B and 8B, the prompted models also produce the state reliably, so the result is not explained by failure to follow the requested format.

Setup. We ask the released Qwen3 models, without fine-tuning, to write SALT's five-field state before every action. The system prompt uses the same field definitions and update rules as the annotation pipeline, together with the writer's inclusion guidance, the output format, and one worked state. The state remains in the agent's own context, is hidden from the user simulator, and begins from the same initial state used by SALT. We run the prompted model once on all 278 τ^2-bench tasks and compare it task by task with the Base and SALT results in Table 2.

The prompt. The prompt given to the untrained models reuses the same field definitions, field rules, and inclusion guidance used to construct the training states (Figure 12). The only substantive change is that the inference-time model must write the pending tool call itself: nothing appends it after generation.

⟨*the sentence above*⟩

```
The <state> block is your private working memory:  the customer never sees it, and it is
kept in your context, so on every turn you update the previous block rather than start
over.  What each line holds:
```

⟨*the five field definitions, Figure 6*⟩

```
How each line changes from turn to turn:
```

⟨*the five field rules, Figure 6, without P's rule "Never a tool call with arguments — the write is added for you."*⟩

⟨*the inclusion sentence below, then the writer's paragraph on the three ways to earn a place, Figure 7*⟩

```
Format:  exactly five lines, G: C: Q: P: E: in that order, one line each.  C, Q and E are
lists whose items are joined by " | "; write "none" when a list is empty.  When your
action this turn is a tool call that changes the customer's records, end P with " |
write:  tool_name(arg=value <- source; ...)" naming where each argument value came from
(the customer, or the tool that returned it).  The block comes first in every reply,
before your message or your tool call.

Example of one block, on the turn an order's address is changed:
<state>
G: change the delivery address on my furniture order - the big one for the living room,
to my work | correction:  I meant the really tall black one, the 6 foot one.  That's the
one for the living room.  The metal one.
C: customer confirmed address change details for order #W2607425 (new address:  500
Commerce Street, Suite 412, Austin, Texas 78701) | policy:  modify pending order shipping
address requires status 'pending' and explicit user confirmation — order #W2607425 status
is pending, customer confirmed yes
Q: order #W2607425 status:  pending
P: execute the address modification on order #W2607425 to 500 Commerce Street, Suite 412,
Austin, Texas 78701 | write:  modify_pending_order_address(zip="78701" <- customer;
country="USA" <- get_order_details; address2="Suite 412" <- customer; city="Austin" <-
customer; address1="500 Commerce Street" <- customer; state="TX" <- get_order_details;
order_id="#W2607425" <- get_order_details)
E: none
</state>
```

The inclusion sentence

```
An item belongs in the state only if the agent, reading the transcript it already has,
would have to WORK to get it — or would MISS it.
```

Figure 12: System-prompt addition for the prompted-state baseline. The abbreviated segments are taken from the same field specification and writer guidance used to construct SALT's training states (Figures 6 and 7).

Prompting does not close the gap. SALT outperforms the prompted base model at every size (Table 8). At 1.7B, 4B, and 8B, SALT reaches 55.9, 69.1, and 72.7 pass^1, compared with 17.8, 38.2, and 42.5 for the prompted models: gains of 38.1, 30.9, and 30.2 points. In contrast, adding the state prompt to the base model changes pass^1 by only -2.8, $+1.5$, and -1.4 points ($20.6 \rightarrow 17.8$,

$36.7 \rightarrow 38.2$, and $43.9 \rightarrow 42.5$). Thu,s the explicit state representation alone does not account for SALT's improvement; the gain appears only when constructing the state is part of training.

	τ^2-bench pass^1			Change (points)	
Model	Base	Prompted	SALT	Prompted − Base	SALT − Prompted
Qwen3-1.7B	20.6	17.8	55.9	−2.8	**+38.1**
Qwen3-4B	36.7	38.2	69.1	+1.5	**+30.9**
Qwen3-8B	43.9	42.5	72.7	−1.4	**+30.2**

Table 8: **Prompting the base model to write the state is not enough; the model must be trained to do so.** pass^1 on τ^2-bench, the unweighted mean over its three domains; higher is better. Base and SALT are the four-trial runs of Table 2; Prompted is the same base model asked by its system prompt to write SALT's five-field state before every reply, run once on all 278 tasks.

The prompted models do write the state. At 4B and 8B, the prompted models emit a state on 82% and 76% of turns, respectively, and nearly all emitted states contain the five fields in the required order. The absence of a gain at these sizes therefore cannot be explained simply by the model refusing or failing to produce the requested structure. The 1.7B model follows the format less reliably, emitting a state on 42% of turns and a well-formed state on 26%. Together with the performance results, this separates *writing a state-shaped object* from learning to construct a useful task state and act from it. Section 6 provides direct evidence for the latter in SALT.

Why prompting is not enough. The τ^2-bench tasks require the state to be assembled from evidence that arrives through different parts of the interaction. The user states and revises the goal and may confirm or withhold consent; tools return records and outcomes that later actions depend on; domain constraints determine whether those facts permit the next action. In telecom, some relevant changes also occur through actions taken by the user outside the agent's tools and must be recovered through dialogue. A prompt can specify the five fields, but it does not provide supervision for resolving these dependencies correctly over a trajectory.

These results do not identify a single cause of the failures, but they show that providing the state schema at inference is not sufficient. SALT instead trains the model on how the state should be constructed from the interaction before the next action is chosen.

D SALT AMONG EXPLICIT-STATE AGENT METHODS

This section expands the state-based-agent comparison in Section 2. Table 9 summarizes the methods discussed there by what their state represents, how it is produced, and how it enters action selection. The discussion below then isolates the differences most relevant to SALT's design and to the controlled comparisons in this paper.

Ontology slots versus a state organized by decision role. SimpleTOD (Hosseini-Asl et al., 2022) and UBAR (Yang et al., 2021) also generate an explicit intermediate representation before the response, but their belief state is defined by a task ontology of domain–slot–value triples. SALT instead organizes information by the role it plays in the next action: goal, constraints, facts, plan, and open issues. The difference is visible in Figure 11: the order's changed status is a fact (Q_t), the one-return-or-exchange condition is a constraint (C_t), and the missing target size is an open issue (E_t). These distinctions are not different slot values; they specify how each item should affect the next decision. Section 6.2 tests whether this organization matters: on WebShop at 4B, neither SALT-flat, which removes the field structure from the same state items, nor a free-form rationale improves over SFT, while SALT does.

Inference-time state maintenance versus state supervision. StateAct (Rozanov & Rei, 2025), IDSS (Tao et al., 2026), and LedgerAgent (Uddin et al., 2026) make state explicit at inference, through prompting or external state-management logic. SALT asks a different question: whether constructing the state should itself be a supervised training target for the acting model. Appendix C provides a direct control. When the released Qwen3 models are prompted to write SALT's five-field

Table 9: Explicit-state methods discussed in Section 2. "Full history" means that prior user messages, actions, and tool results remain available when selecting the next action. Descriptions summarize each method's stated representation and training or inference procedure, rather than every fact its state could potentially encode.

Method	What the state records	How the state is produced and used
SALT (ours)	Goal, applicable constraints, established facts, remaining plan, and unresolved issues at each action point, organized by their role in the next decision.	States are derived automatically from existing trajectories using only evidence available before the action. The acting model is trained to write the state and then the demonstrated action. The full interaction history remains available.
SimpleTOD (Hosseini-Asl et al., 2022)	Dialogue belief as domain–slot–value triples defined by the task ontology, followed by database results, dialogue act, and response.	A single autoregressive model learns belief, act, and response from annotated task-oriented dialogue.
UBAR (Yang et al., 2021)	Domain–slot–value belief state over the dialogue ontology.	A single model generates belief, act, and response over the session from annotated dialogue trajectories.
StateAct (Rozanov & Rei, 2025)	A prompted representation of the task goal and evolving environment state, such as location and inventory.	Maintained through prompting at inference; no state-supervised fine-tuning of the acting model.
IDSS (Tao et al., 2026)	A two-layer situation state containing provenance-aware entities and attributes together with user intents, required variables, constraints, and execution status.	Constructed at inference without training; the state is rendered back to the model together with residual interaction context.
LedgerAgent (Uddin et al., 2026)	A ledger of records and task facts established by successful tool returns, together with the conditions checked before environment-changing actions.	Maintained deterministically outside the model at inference and rendered into the prompt; a gate checks proposed writes against the ledger and domain constraints.
WKM (Qiao et al., 2025)	Global task knowledge and local state knowledge summarized from expert trajectories.	A separate knowledge model produces the state; the acting model is trained on knowledge-augmented trajectories and consumes that state.
ABBEL (Lidayan et al., 2026)	A recursively updated natural-language belief over task-relevant unknowns.	Belief content is optimized with RL-based signals and replaces earlier interaction history when the next action is selected.
PABU (Jiang et al., 2026)	Current progress together with selected earlier actions, observations, and available actions.	Supervises progress and retention decisions alongside action prediction; the policy acts from the retained belief rather than the full history.
PARL-MT (Chai et al., 2025)	A compact representation of interaction progress, including a history summary, future plan, and rationale.	Awareness targets are synthesized and verified, used for SFT warm-up, then followed by progress-guided RL.
IACM-RL (Zhu et al., 2026)	Confirmed parameter values, stale-value indicators, current goal, recent tool execution, pending clarifications, and an intent-shift signal.	The policy generates the block before acting and is trained with RL and auxiliary objectives; the resulting behavior is later distilled into a student that acts without the explicit block.

state without state-augmented training, prompting does not improve pass^1 at any model size, while SALT remains 30–38 points higher. At 4B and 8B the prompted models produce the requested structure on most turns, separating the ability to write a state-shaped object from learning to construct a useful state and act from it.

A separate state model versus one model for state and action. WKM (Qiao et al., 2025) obtains task knowledge from a separate model that the agent then consumes. SALT instead trains the acting model itself to generate the state immediately before the action (Equation (2)), with no additional model at inference (Section 3.4). This coupling matters empirically: Section 6.2 edits an identifier in the generated state while leaving the correct identifier in the conversation; in over 70% of those cases, the regenerated action follows the edited state. The state is therefore part of the policy's decision process rather than an auxiliary annotation produced alongside it.

Replacing the history versus supplementing it. ABBEL (Lidayan et al., 2026) uses a learned belief in place of earlier interaction history, so that representation must carry the information needed for subsequent action selection. SALT keeps the full history and uses the task state to surface information that would otherwise have to be recovered from it. The state therefore complements rather than replaces the interaction. The additional state tokens increase total tokens processed by only 2–14% on τ^2-bench, while SALT uses fewer agent turns and tool calls (Table 5).

Partial state objectives versus a shared task-state target. PABU (Jiang et al., 2026), PARL-MT (Chai et al., 2025), and IACM-RL (Zhu et al., 2026) supervise or optimize representations that overlap with parts of SALT's state, including progress, retained observations, plans, confirmed values, and unresolved information. SALT combines these roles in one shared target and additionally makes the currently applicable constraints an explicit field. The methods also differ in how their intermediate targets are constructed. PARL-MT synthesizes awareness using the demonstrated call and future interaction, whereas SALT's annotation at turn t is restricted to evidence available before action a_t (Section 3.3). PABU can modify some action targets during training; SALT preserves the demonstrated actions and adds the task state before them. Removing the states therefore recovers the original trajectory, making the SFT comparison isolate the additional task-state supervision as closely as possible.

These methods differ in benchmark, data, base model, and training procedure, so their reported scores are not directly comparable. The purpose of Table 9 is instead to locate SALT along three design axes: what the state represents, whether the acting model is trained to construct it, and whether the state replaces or supplements the interaction history.