EXPMEM: MITIGATING HALLUCINATIONS IN LLM AGENTS VIA EXPECTATION-DRIVEN MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) inherit extensive general priors from pre-training to support reasoning and decision-making, yet these generic priors often conflict with the specific rules and operational constraints of real deployment environments. Such mismatches lead to severe action-feasibility hallucinations, invalid agent behaviors, and stalled task progress. Existing post-training remedies, including supervised fine-tuning and reinforcement learning, can alleviate such issues but require costly data collection and computational overhead. To tackle this problem, we propose ExpMem, an expectation-driven memory framework that mitigates hallucinations in LLM agents. Our framework builds compact, structured environmental memory to encode heterogeneous environmental rules, and equips a memory-augmented reasoning module to retrieve constraint-aware knowledge and fuse it into real-time decision-making. This design accelerates environmental exploration, eliminates prior-induced reasoning biases, and effectively suppresses mismatch-caused hallucinations. Benefiting from its parameter-free and plug-and-play nature, our method provides a low-cost, adaptable solution for real-world LLM agent deployment. Experiments on ALFWorld and ScienceWorld show that ExpMem reduces repeated ineffective decisions while improving task success and interaction efficiency. Code is available at: `https://anonymous.4open.science/r/expmem/`

1 INTRODUCTION

Large language models (LLMs) (Achiam et al., 2023; Liu et al., 2024) acquire extensive general knowledge during pretraining, forming priors that guide reasoning and decision-making. However, these priors may diverge substantially from the rules and operational constraints of real deployment environments (Wu et al., 2024; Yao et al., 2024). Such mismatches can induce hallucinations about action feasibility, leading to ineffective actions and stalled task progress (Ahn et al., 2022). For example, an LLM-controlled delivery robot may assume that pressing an elevator's floor button is sufficient to reach its destination, overlooking the building-specific requirement for access-card authentication. Although post-training methods such as supervised fine-tuning (SFT) (Zeng et al., 2024) and reinforcement learning (RL) (Kaelbling et al., 1996; Carta et al., 2023) can help mitigate such hallucinations, they incur substantial data collection and computational costs, which are often unaffordable in real-world applications.

We aim to design an effective memory framework that mitigates mismatches between LLM priors and environmental specifications without updating model parameters. This problem presents three key challenges. First, discrepancies between LLM priors and environmental specifications are often implicit, making them difficult to identify and articulate in memory by natural language (Shinn et al., 2023). Second, under a limited number of interactions, agents often struggle to acquire enough knowledge of the environmental specifications (Majumder et al., 2023). Accordingly, the desired memory framework needs to maintain a compact representation and accelerate exploration of the environment. Third, even when agents recognize discrepancies between their priors and the environmental specifications, they may still fail to utilize this knowledge during the reasoning process, thus causing incorrect actions.

Existing memory-augmented agents reuse interaction trajectories or summarize experience to guide subsequent decision-making. Synapse (Zheng et al., 2024) retrieves abstracted trajectories as

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.

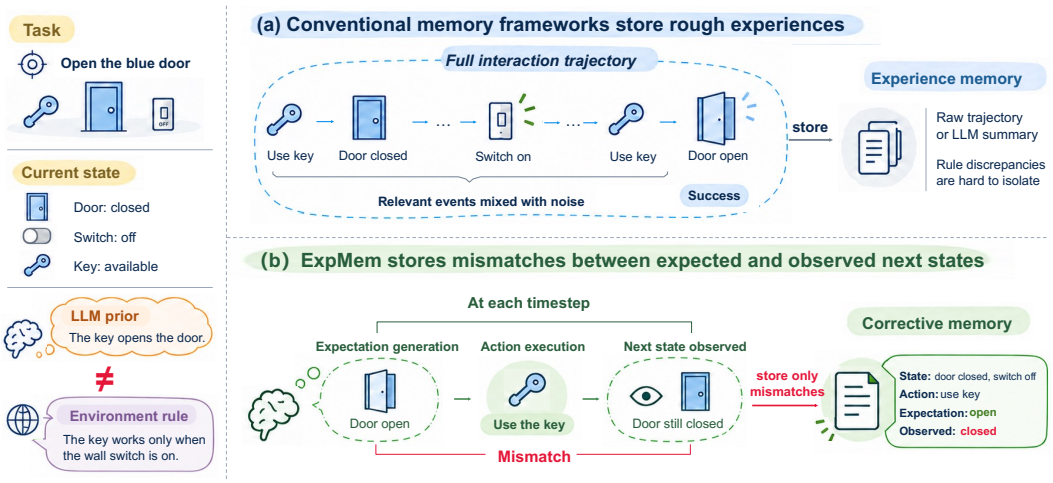


Figure 1: **Motivation for ExpMem.** (a) Storing trajectories or broad summaries may leave mismatches between LLM priors and environmental specifications unidentified. (b) ExpMem compares intended and observed next states to identify mismatches and construct structured memories.

demonstrations, while Reflexion (Shinn et al., 2023) and ExpeL (Zhao et al., 2024) extract verbal reflections or reusable insights from completed attempts. However, even successful trajectories may contain local failures interspersed with unrelated exploration and redundant steps. As illustrated in Figure 1(a), evidence of mismatches between LLM priors and environmental specifications may remain implicit in stored trajectories or experience summaries. WALL-E (Zhou et al., 2026) and WorldMind (Ren et al., 2026) acquire knowledge of environmental specifications from interaction feedback to improve state prediction and planning. However, constructing and updating this knowledge requires additional LLM calls, increasing the computational cost of adaptation. These limitations motivate a memory framework that efficiently identifies and retains evidence of such mismatches and uses it to improve subsequent decision-making.

To address these challenges, we propose **ExpMem**, an **Exp**ectation-driven **Mem**ory framework for mitigating hallucinations in LLM agents without updating model parameters. Its core idea is to represent the agent's expectations as an expected next state that supports task progress and is judged achievable under the current environmental conditions, and to use interaction feedback to refine these judgments through memory. ExpMem first specifies an expected next state based on the task goal, current state, available actions, and retrieved memories, and then selects an action to achieve it. After execution, ExpMem compares the expected next state with the observed next state to identify mismatches. A mismatch indicates that the executed action did not achieve the expected next state under the current conditions. Predefined code rules construct structured memories from these mismatches without additional LLM calls for memory construction. These memories are available for retrieval from the next step, enabling their reuse within the current task. Using these memories to revise the expected next state and inform subsequent action selection aims to reduce repeated ineffective attempts and make better use of limited interactions for exploration. Our main contributions are summarized as follows.

- We introduce an expectation-guided decision process that explicitly generates an expected next-state prior to action selection. Comparing this expected state against the observed environmental outcome enables systematic detection of prior-environment mismatches.
- We present ExpMem, a parameter-free expectation-driven memory framework. Using predefined code rules rather than additional LLM calls, ExpMem constructs structured four-component memory records from detected mismatches.
- We conduct rigorous evaluations on two interactive benchmarks (ALFWorld, ScienceWorld) across two distinct LLM backbones. Empirical results show ExpMem yields higher task success rates, reduces repetitive ineffective interaction loops, and makes more efficient use of limited interaction budgets compared to strong memory-augmented baselines.

2 Related Work

Grounding LLM Agents in Interactive Environments. Training-based approaches, including supervised fine-tuning and reinforcement learning, can help ground LLM agents' decision-making in environmental specifications. AgentTuning (Zeng et al., 2024) combines interaction trajectories with general instruction data to strengthen agent capabilities while preserving general language abilities. Agent-FLAN (Chen et al., 2024) disentangles format following from agent reasoning and uses negative examples to alleviate hallucinations. AgentGym (Xi et al., 2025) iteratively fine-tunes agents on successful trajectories across diverse environments, while Agent-R (Yuan et al., 2025) learns error recovery from self-generated revision trajectories. Beyond learning from demonstrations, reinforcement learning uses environmental feedback to optimize agents' behavior over successive interactions. GLAM (Carta et al., 2023) grounds LLM policies through online reinforcement learning. RAGEN (Wang et al., 2025) studies multi-turn reinforcement learning through trajectory-level policy optimization, focusing on rollout design and training stability. These approaches incorporate experience into model parameters, requiring training data or environment rollouts as well as additional computation for optimization. ExpMem focuses on adaptation during task execution through an external memory that retains evidence of mismatches between intended and observed outcomes. By reusing this feedback to inform subsequent decisions, ExpMem helps agents adjust to environmental specifications without updating model parameters.

Memory-Augmented LLM Agents. Prior work retains interaction experience in several forms, including trajectory exemplars in Synapse (Zheng et al., 2024); verbal reflections and reusable insights in Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), and ReasoningBank (Ouyang et al., 2026); executable skills in Voyager (Wang et al., 2023); and reusable workflows in AWM (Wang et al., 2024). Other methods improve memory organization and retrieval: A-MEM (Xu et al., 2026) dynamically links and updates memory notes, CDMem (Gao et al., 2025) organizes experience using task and environmental context, and MemRL (Zhang et al., 2026) learns utility scores from environmental feedback to guide retrieval. These methods emphasize experience reuse and memory organization. Our focus is on how feedback about mismatches between LLM priors and environmental specifications is retained and used to revise subsequent decisions. WALL-E (Zhou et al., 2026) and WorldMind (Ren et al., 2026) learn environmental knowledge from interaction feedback to improve predictions of action outcomes. ExpMem instead uses memory to determine what progress is feasible next, guiding the generation of expected next states that advance the task while satisfying environmental specifications. By revising unsupported expectations before action selection, ExpMem helps avoid repeated ineffective attempts and directs limited interactions toward feasible task progress.

3 Preliminaries

Memory-Augmented LLM Agents. We consider a goal-conditioned LLM agent interacting with an environment over multiple steps. At step t, the agent receives an observation o_t and maintains an interaction history $h_t = (o_0, a_0, \ldots, o_t)$ for a task goal g. A memory-augmented agent additionally maintains an external memory $\mathcal{M}_t$ containing information acquired from previous interactions. Given the current decision context, it constructs a query q_t and retrieves relevant memory entries: $m_t = \mathcal{R}(q_t, \mathcal{M}_t)$, where $\mathcal{R}$ denotes the retrieval function. The retrieved memories are incorporated into the prompt together with the task goal and interaction history to form the current context, $c_t = \mathcal{C}(g, h_t, m_t)$, from which the agent generates an action: $a_t \sim \pi_\theta(\cdot \mid c_t)$. After executing a_t and receiving the next observation o_{t+1}, the agent extracts interaction evidence ξ_t and updates its memory as $\mathcal{M}_{t+1} = \mathcal{U}(\mathcal{M}_t, \xi_t)$, where $\mathcal{U}$ is the memory update function. The model parameters θ remain fixed during interaction; thus, test-time adaptation is achieved by updating the external memory and the resulting decision context rather than the model parameters. Existing memory mechanisms primarily differ in how interaction evidence ξ_t is represented, retrieved, and updated.

4 Method

Motivation. During task execution, an agent selects actions with an implicit expectation of how the environment will change. An expected change may appear useful for the task yet fail to occur under the current conditions. Making this expectation explicit allows the agent to compare the intended

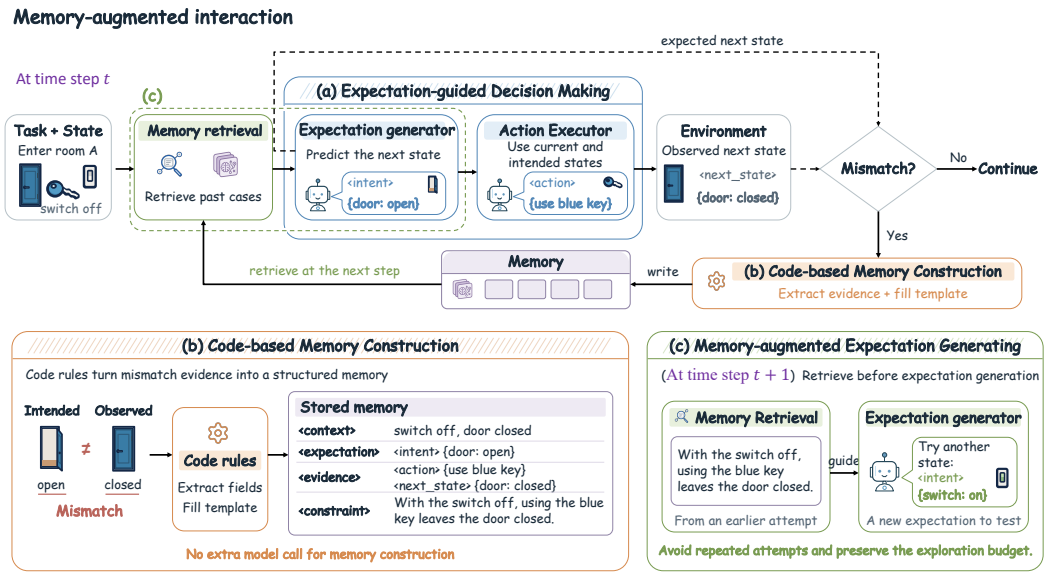


Figure 2: Overview of ExpMem. (a) The expectation generator predicts the expected next state, and the Action Executor generates an action based on the current and intended states. (b) Code rules convert mismatches between intended and observed next states into structured memories containing context, expectation, evidence, and constraint, without additional model calls. (c) Retrieved memories guide subsequent expectation prediction, helping the agent avoid repeated unsuccessful attempts and preserve its exploration budget.

and observed next states and identify which part of its expectation was not realized. Retaining these discrepancies together with their conditions and evidence allows past interactions to inform later expectation generating. The aim is to help the agent reconsider unsupported expectations when similar situations recur and direct its exploration toward other ways of making progress.

Overview. We introduce ExpMem, which connects expectation-guided decision making, code-based memory construction, and memory-augmented planning (Figure 2). The expectation generator first proposes an expected next state from the current state, task goal, and retrieved memories. The Action Executor then selects an action to attempt this transition. Code rules compare the intended outcome with environmental evidence, extract supported discrepancies, and populate a structured memory template. These records are retrieved before subsequent planning steps, allowing newly acquired evidence to influence the next intended state while the task is still in progress.

4.1 Expectation-guided Decision Making

At each time step, ExpMem first predicts the next state it aims to reach, then generates an action based on the current state and this prediction. This prediction guides action generation and provides a reference for checking the observed next state, linking local decision making to memory construction. At step t, let s_t denote the current state representation constructed from observations and history, g the task goal, and $\mathcal{R}_t$ the retrieved memories.

Expectation generator. The expectation generator combines these inputs to generate an expected next state:

$$\hat{s}_{t+1} = \mathcal{P}(s_t, g, \mathcal{R}_t). \tag{1}$$

Here, $\hat{s}_{t+1}$ denotes the expected next state that the expectation generator expects the agent to reach after the next action. The prediction includes only the state information relevant to the current expectation. For example, when the task is to enter a room, the Planner may predict *door: open*, without predicting other parts of the environment state. The expected next state may directly advance the task or satisfy a prerequisite for a later step. The Planner uses the retrieved memories $\mathcal{R}_t$ when predicting this next state, as detailed in Section 4.3.

Action generation. The Action Executor receives the current state s_t and the expected next state $\hat{s}_{t+1}$, and generates an action that attempts to realize the specified transition:

$$a_t = \mathcal{E}(s_t, \hat{s}_{t+1}). \tag{2}$$

The Planner and Executor are implemented as two sequential language-model calls. After executing the action, the agent obtains the observed next state s_{t+1} from environmental feedback. We compare the expected next state with the observed next state to identify mismatches, which provide evidence for memory construction.

4.2 Code-based Memory Construction and Evolution

ExpMem uses code rules to detect mismatches, extract evidence, and fill memory templates. New records are created only when environmental feedback or the current legal action set provides explicit evidence of a mismatch. Memory construction requires no additional language-model calls.

Mismatch detection and evidence extraction. After execution, code rules compare the intended next state $\hat{s}_{t+1}$ with the observed next state s_{t+1} and inspect the environmental feedback to identify mismatches. The comparison checks whether the expected next state was reached. For each mismatch supported by the feedback, the rules extract the conditions of the attempt, the executed action, and the observed next state for memory construction.

When the environment provides a legal action set, code rules can also check whether an action required for the expected next state is available. This check uses the legal actions available before execution. If the required action is absent, the rules retain this evidence for memory construction at the end of the step. The resulting memory records that the action was unavailable, without describing it as an action that was executed and failed.

Four-part memory representation. We describe the information stored in each memory using four components:

$$m_i = \langle c_i, e_i, o_i, g_i \rangle. \tag{3}$$

These components connect the conditions of a mismatch, the expected next state, the evidence of the mismatch, and its use in subsequent planning.

Context (c_i). The state and task conditions when the intended next state was predicted. These conditions help the Planner determine when the recorded evidence applies.

Expectation (e_i). The expected next state associated with the mismatch. This component identifies the prediction to reconsider in subsequent planning.

Evidence (o_i). The executed action and environmental feedback showing the mismatch, or the legal-action evidence showing that a required action was unavailable. This component preserves the evidence used by the code rule.

Constraint (k_i). A statement generated by a fixed template from the recorded conditions and evidence to guide subsequent expectation prediction. For an executed attempt, it states what the recorded action failed to achieve under those conditions; it does not rule out reaching the same expected next state through a different action.

As shown in Figure 2, *Context* records an off switch and a closed blue door, *Expectation* specifies *door: open*, and *Evidence* records *use blue key* followed by *door: closed*. The resulting *Constraint* states: "With the switch off, using the blue key leaves the door closed." The memory therefore connects this unsuccessful attempt to its recorded conditions for use in subsequent planning.

Together, these components connect the circumstances of an unmet expectation, the observed basis for revising it, and the implication for future planning. In the main figure's example, *Context* records an off switch and a closed blue door; *Expectation* specifies an open door; *Evidence* records that using the blue key left the door closed; and *Constraint* advises against expecting that action to open the door under the same conditions. This record does not establish the switch as the sole cause of failure or specify an untested action as a verified remedy.

Online writing. New memories are written at the end of the current step and become available for retrieval from the next step, without waiting for task completion or a final reward. Later successful feedback can update an existing memory with newly verified evidence. Fixed extraction rules and

templates make memory construction reproducible for the same inputs, while the retained evidence allows each memory to be traced back to the interaction that produced it.

4.3 Memory Use in Decision Making

Retrieval before planning. Let $\mathcal{M}_t$ denote the memory bank at the beginning of step t. Before the expectation generator predicts the expected next state, ExpMem retrieves memories using the current state and task goal:

$$\mathcal{R}_t = \text{Retrieve}(\mathcal{M}_t; s_t, g). \tag{4}$$

Code rules filter memories by reliability and check their recorded conditions against the current state. The remaining memories are ranked using a fixed weighted score based on their match with s_t and g, together with their reliability. The top-k memories are returned, or all remaining memories if fewer than k are available.

Expectation generation with retrieved memory. The retrieved memories $\mathcal{R}_t$ are provided as text to the expectation generator alongside s_t and g. The Planner uses the recorded conditions and evidence to reconsider expected next states associated with earlier mismatches. When memory conflicts with current observations or legal actions, the Planner is instructed to prioritize the current information.

Action generation based on expectation. As illustrated in Figure 2, the retrieved memory records that using the blue key with the switch off left the door closed. This evidence can guide the expectation generator at time step $t + 1$ to predict *switch: on* as a new expected next state, avoiding another key-use attempt under the same conditions. The Action Executor then generates an action based on the current state and this expected next state. ExpMem thus uses memory to guide expectation prediction without updating model parameters, aiming to reduce repeated unsuccessful attempts and preserve the exploration budget for other attempts that may advance the task.

5 Experiments

We evaluate ExpMem through the following research questions:

- **RQ1:** How does ExpMem compare with existing baselines in cumulative task success?
- **RQ2:** How do memory use, retention, content and construction relate to ExpMem's performance?
- **RQ3:** How effectively does ExpMem reduce unproductive interaction loops during task execution?

5.1 Experimental Setup

Environments and backbones. We use AgentBoard (Ma et al., 2024) to evaluate ExpMem on ALFWorld (Shridhar et al., 2020) and ScienceWorld (Wang et al., 2022), with 134 and 150 tasks, respectively. ALFWorld requires object search and household manipulation, whereas ScienceWorld involves scientific reasoning and multistep experimental procedures. We evaluate both environments with MiniMax-M3 and Qwen-3.7-Max, allowing us to examine performance across different task domains and model backbones.

Baselines. We compare ExpMem with eight baselines, grouped by their primary approach to reasoning and memory use:

- **Reasoning and reflection:** ReAct (Yao et al., 2022) interleaves reasoning and actions, while Reflexion (Shinn et al., 2023) retains verbal feedback to guide later attempts.
- **General-purpose memory:** Mem0 (Chhikara et al., 2025) extracts and consolidates interaction memories, and A-MEM (Xu et al., 2026) organizes linked memory notes.
- **Memory for interactive decision-making:** SAMem (Wang et al., 2026) retrieves memories according to the agent's state. AdaMEM (Yan et al., 2026) reuses strategies derived from successful trajectories to guide subsequent decisions. CDMem (Gao et al., 2025) organizes and retrieves task and environment knowledge according to the interaction context.

Table 1: Cumulative success (%, SR@r). Best and second-best scores are **bold** and underlined. Red values give gains (pp) over the best baseline in each column and backbone.

		ALFWorld (N=134)			ScienceWorld (N=150)		
Type	Method	SR@1	SR@2	SR@3	SR@1	SR@2	SR@3
MiniMax-M3							
Reasoning	ReAct	70.90	76.87	82.09	25.33	37.33	46.67
Reasoning	Reflexion	67.91	82.84	86.57	28.00	38.67	46.67
Memory	Mem0	69.40	79.85	80.60	29.33	40.00	46.00
Memory	A-MEM	71.64	82.84	85.82	28.67	40.67	46.67
Memory	SAMem	70.90	83.58	86.57	30.00	44.00	52.00
Memory	AdaMEM	70.90	78.36	82.09	30.00	45.33	56.67
Memory	CDMem	69.40	79.10	84.33	26.00	43.33	51.33
Memory	WorldMind	71.64	76.87	79.85	32.00	42.67	45.33
Memory	**ExpMem**	**85.82** +14.18	**93.28** +9.70	**97.76** +11.19	**55.33** +23.33	**66.00** +20.67	**73.33** +16.67
Qwen-3.7-Max							
Reasoning	ReAct	70.90	75.37	77.61	20.00	22.67	26.00
Reasoning	Reflexion	69.40	79.85	82.84	16.00	24.67	28.00
Memory	Mem0	70.15	74.63	75.37	21.33	26.67	28.00
Memory	A-MEM	79.10	84.33	87.31	23.33	26.67	29.33
Memory	SAMem	81.34	86.57	91.79	32.00	40.67	42.00
Memory	AdaMEM	67.91	82.09	84.33	16.00	26.67	30.00
Memory	CDMem	64.18	85.82	88.06	25.33	35.33	40.67
Memory	WorldMind	72.39	75.37	77.61	26.00	33.33	35.33
Memory	**ExpMem**	**91.04** +9.70	**97.01** +10.45	**97.01** +5.22	**51.33** +19.33	**61.33** +20.67	**63.33** +21.33

- **World-knowledge memory:** WorldMind (Ren et al., 2026) learns from prediction errors and successful trajectories.

5.2 Overall Performance (RQ1)

Evaluation protocol and metrics. Each task receives at most three attempts. All tasks are evaluated in the first round; subsequent rounds retry only unsolved tasks. ExpMem retains task-specific memory across attempts and allows eligible environment-level rules to be reused across tasks. We report cumulative success, SR@r, defined as the percentage of the original task set solved within r attempts, where $r \in \{1, 2, 3\}$. The denominator remains fixed across rounds. To assess efficiency, we additionally compare the cumulative main interaction steps and recorded model tokens across all executed attempts, serving as proxies for interaction and inference costs, respectively.

Overall task success. Table 1 shows that ExpMem achieves the highest success rate in every round across both environments and backbones, with final gains of 5.22–21.33 percentage points over the strongest baseline in each setting. On ALFWorld, ExpMem reduces the number of unresolved tasks from 18 to 3 with MiniMax and from 11 to 4 with Qwen, corresponding to reductions of 83.3% and 63.6%, respectively. These reductions highlight the value of the improvements near the performance ceiling, where relatively small gains in success rate can represent a substantial reduction in remaining failures. On ScienceWorld, ExpMem exceeds the strongest baselines, AdaMEM with MiniMax and SAMem with Qwen, by 16.67 and 21.33 percentage points, respectively. These results show that ExpMem consistently improves task success across the evaluated environments and backbones, extending task coverage even when the strongest baselines already achieve high success rates.

Performance across attempts. ExpMem's advantage is already evident under a limited attempt budget. On ALFWorld, its first-attempt success falls only one task (0.75 percentage points) short of the strongest baseline's three-attempt result with either MiniMax or Qwen. On ScienceWorld, MiniMax comes within 1.33 points of that reference, while Qwen already exceeds it by 9.33 points. After two attempts, ExpMem surpasses the strongest baseline's three-attempt success in all four settings, with margins ranging from 5.22 to 19.33 points. Thus, ExpMem achieves higher task success with fewer retry opportunities and maintains its advantage when all methods are allowed the full three-attempt budget.

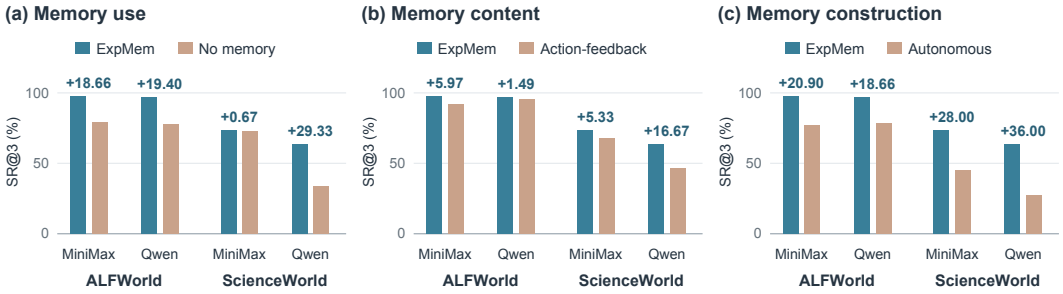


Figure 3: Memory mechanism analysis: (a) memory use, (b) memory content, and (c) memory construction and use. Bars show SR@3 (%); labels indicate ExpMem's gains (pp).

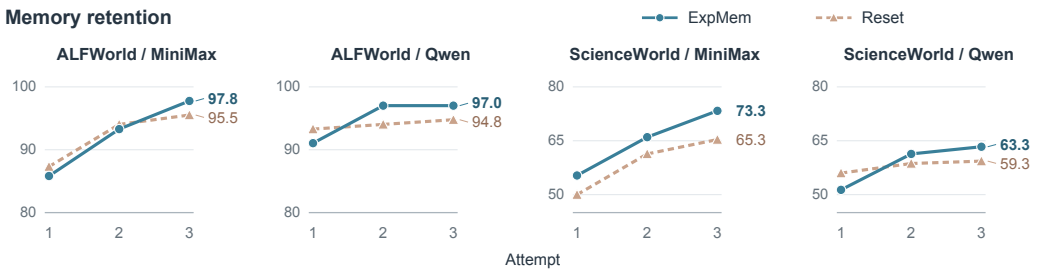


Figure 4: Memory retention analysis. Curves show cumulative success (%) across attempts; labels indicate final scores. Vertical scales differ by environment.

5.3 Memory Mechanism Analysis (RQ2)

Ablation settings. We analyze ExpMem's memory mechanism along four dimensions: *memory use*, *memory retention*, *memory content*, and *memory construction and use*. All configurations follow the three-attempt protocol in Section 5.2. (1) For *memory use*, No memory disables writing, retrieval, and memory-derived planning constraints. (2) For *memory retention*, Reset between attempts uses memory within each attempt but clears it before the next attempt, preventing reuse across attempts or tasks. (3) For *memory content*, Action–feedback only preserves memory-write triggers, merging, retrieval, and lifetime, but replaces explicit expectations, mismatch interpretations, and planning guidance with raw action–feedback records. These three variants retain the expectation-planning and action-execution framework. (4) For *memory construction*, Autonomous experience directly generates actions and uses an LLM to select action–feedback records for storage and retrieval, without explicit expectation prediction.

(1) Memory use. Memory improves SR@3 by 18.66 and 19.40 percentage points on ALFWorld with MiniMax and Qwen, respectively, and by 29.33 points on ScienceWorld with Qwen (Figure 3a). In all three settings, ExpMem's first-attempt success already exceeds No memory's three-attempt result. MiniMax shows a smaller gain on ScienceWorld, where the underlying framework already achieves 72.67% without memory. These results show that making accumulated evidence available to planning can provide gains that additional retries alone do not recover, complementing the underlying planning capability.

(2) Memory retention. Retaining memory improves final success in all four settings (Figure 4): by 2.24 points with both backbones on ALFWorld, and by 8.00 and 4.00 points with MiniMax and Qwen on ScienceWorld. With Qwen, ExpMem solves eight additional ALFWorld tasks and eighteen ScienceWorld tasks after the first attempt, compared with two and five for Reset. This pattern is consistent with later attempts building on evidence gathered during earlier interactions. Preserving that evidence allows an unsuccessful execution to remain useful for subsequent planning, supporting stronger performance over repeated attempts.

(3) Memory content. ExpMem outperforms Action–feedback only by 5.97 and 1.49 points on ALFWorld, and by 5.33 and 16.67 points on ScienceWorld, with MiniMax and Qwen, respectively

(Figure 3b). Raw records already outperform No memory on ALFWorld with both backbones and on ScienceWorld with Qwen, showing that selected events retain useful information. However, these records leave the planner to reconstruct which expectation was contradicted and how the observation should change its decisions. ExpMem makes these implications explicit through its memory representation and associated planning guidance. The results favor memory that conveys the planning implications of an experience in addition to the event itself.

(4) Memory construction. ExpMem exceeds Autonomous experience by 20.90 and 18.66 points on ALFWorld and by 28.00 and 36.00 points on ScienceWorld, with MiniMax and Qwen, respectively (Figure 3c). This alternative already selects, stores, and retrieves experiences, yet does not match ExpMem's performance. The end-to-end advantage is consistent with the value of coordinating experience construction with explicit planning. In ScienceWorld, the autonomous agent also encounters execution-validation failures, indicating that effective experience reuse must support executable decisions. Together with the content ablation, these results support a design that connects selected experience to expectations and subsequent planning, beyond the autonomous collection of interaction records.

5.4 Repetitive Loops and Hallucination Mitigation (RQ3)

Persistent errors about action feasibility or effects can lead agents to repeat interactions without progress. We examine whether ExpMem reduces such stagnant repetition across environments and backbones.

Table 2: Loop-step rate (%, lower is better).

Backbone	ExpMem	Best	ReAct
ALFWorld			
MiniMax	**3.66**	7.22	10.30
Qwen	**2.53**	9.88	33.17
ScienceWorld			
MiniMax	**5.17**	14.99	10.06
Qwen	**18.20**	43.73	71.20

Best: baseline with highest SR@3; SAMem except AdaMEM for ScienceWorld/MiniMax.

Loop-step rate. Within each task attempt, we flag a repeated interaction when its normalized observable pre-action state, action, and feedback match an earlier interaction, with no new observable state, location–feedback pair, or score change in between. Only repeated occurrences are counted; first occurrences, explicit waiting, and steps yielding score gains or task success are excluded from the numerator. Repetition histories are reset between attempts. LSR is the percentage of flagged steps among all executed main steps across the three rounds, including waiting and steps from unsuccessful attempts. The same rules are applied to all methods.

Results. ExpMem achieves lower LSR than both the strongest baseline and ReAct in all four settings (Table 2). Relative to the strongest baseline, LSR decreases by 49.3% and 74.4% on ALFWorld, and by 65.5% and 58.4% on ScienceWorld, with MiniMax and Qwen, respectively. Absolute loop-step counts also decrease in every setting. Together with the success-rate improvements, these results show that ExpMem sustains task progress with fewer stagnant repetitions. Interaction and inference costs are analyzed in Appendix A.

6 Conclusion

In this paper, we tackle hallucinations caused by mismatches between LLMs' pre-learned priors and environment-specific rules for interactive agents. We introduce ExpMem, an expectation-driven, parameter-free memory framework that detects mismatches by comparing explicitly predicted expected next states with real environmental observations. Mismatch evidence is compiled into structured memories via predefined code rules without additional LLM calls; these memories are retrieved online during task execution to revise agent expectations and curb repeated ineffective actions. Evaluations on ALFWorld and ScienceWorld with two model backbones show that ExpMem improves task success rates and cuts repetitive interaction loops relative to competitive memory-augmented baselines.

AI USE STATEMENT

In this work, we used generative AI tools in three areas: research ideation or execution, literature retrieval and discovery, and manuscript writing and revision. These tools supported the development of ExpMem, our expectation-driven memory framework for LLM agents; helped identify relevant literature on LLM agents and memory-augmented decision making; and assisted in improving the clarity, organization, and readability of the manuscript.

The authors take full responsibility for the research and its presentation, including the validity of the reported results, the accuracy of citations, and all text, claims, and artifacts developed with AI assistance.

REPRODUCIBILITY STATEMENT

To support reproducibility, Section 5.1 specifies the evaluation environments, task sets, model backbones, and baseline methods. Section 5.2 describes the three-attempt evaluation protocol and cumulative success metrics, while Section 5.3 details the memory ablation configurations. Section 5.4 defines the shared loop-detection rules, normalization procedures, and aggregation convention used for the loop-step rate. The method section describes expectation generation, action execution, and memory construction and retrieval. The appendix further documents the ALFWorld state representation, agent prompts, and an execution trajectory with corresponding memory records, illustrating how the framework operates during interaction.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International conference on machine learning*, pp. 3676–3713. PMLR, 2023.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-flan: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 9354–9366, 2024.

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

Pengyu Gao, Jinming Zhao, Xinyue Chen, and Long Yilin. An efficient context-dependent memory framework for llm-centric agents. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 1055–1069, 2025.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362, 2024.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. Clin: A continually learning language agent for rapid task adaptation and generalization. *arXiv preprint arXiv:2310.10134*, 2023.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Baochang Ren, Yunzhi Yao, Rui Sun, Shuofei Qiao, Ningyu Zhang, and Huajun Chen. Aligning agentic world models via knowledgeable experience learning. *arXiv preprint arXiv:2601.13247*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

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

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

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Tong Wang, Pei Xu, Shiyue Cao, Likun Yang, Daipeng Li, Jianbin Jiao, and Kaiqi Huang. Samem: State-aware memory as a fine-grained memory for llm agents in decision-making. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 14691–14710, 2026.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. In *Proceedings of the 2024 conference of the north american chapter of the association for computational linguistics: Human language technologies (volume 1: Long papers)*, pp. 1819–1862, 2024.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, et al. Agentgym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27914–27961, 2025.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

Shannan Yan, Jingchen Ni, Leqi Zheng, Jiajun Zhang, Peixi Wu, Dacheng Yin, Jing Lyu, Chun Yuan, and Fengyun Rao. Adamem: Adaptive user-centric memory for long-horizon dialogue agents. *arXiv preprint arXiv:2603.16496*, 2026.

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

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

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*, 2025.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026.

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

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *International Conference on Learning Representations*, volume 2024, pp. 19036–19066, 2024.

Siyu Zhou, Tianyi Zhou, Yijun Yang, Guodong Long, Deheng Ye, Jing Jiang, and Chengqi Zhang. Wall-e: World alignment by neurosymbolic learning improves world model-based llm agents. *Advances in Neural Information Processing Systems*, 38:65352–65391, 2026.

A Experiment: Interaction and Inference Costs

On ALFWorld, we compare ExpMem with SAMem, which attains the highest baseline SR@3 under both backbones. We report task success alongside cumulative interaction steps and model tokens across all evaluated attempts.

Interaction cost. able 3 shows that ExpMem improves SR@3 by 11.19 and 5.22 percentage points with MiniMax and Qwen, respectively, while reducing total interaction steps by 38.5% and 37.5%. In absolute terms, ExpMem completes 15 additional tasks with MiniMax and 7 with Qwen using fewer interactions across the entire task set. When cumulative steps across all attempts are divided by the number of completed tasks, the interaction cost per completed task decreases by 45.6% and 40.9%, respectively. Thus, ExpMem's advantage holds both in total interaction expenditure and in expenditure relative to completed work. These results demonstrate that ExpMem achieves broader task coverage while making more effective use of environment interactions under both backbones.

Table 3: Task success and cumulative costs on ALFWorld.

Model	Method	SR@3 (%)	Steps	Tokens (M)
MiniMax	SAMem	86.57	3,865	8.08
MiniMax	ExpMem	**97.76**	**2,376**	**7.24**
Qwen	SAMem	91.79	3,036	**6.69**
Qwen	ExpMem	**97.01**	**1,898**	7.34

Inference cost. ExpMem achieves higher task success while keeping total token expenditure close to that of SAMem. With MiniMax, it completes 15 additional tasks while reducing token consumption by 10.4%, improving task success and reducing both interaction and inference costs simultaneously. With Qwen, 7 additional task completions accompany a 9.8% increase in total tokens. Using the same normalization—total tokens across all attempts divided by the number of completed tasks—the inference cost per completed task decreases by 20.7% with MiniMax and increases by only 3.9% with Qwen. The additional task coverage therefore comes with lower or nearly unchanged token expenditure per completed task. Together with the substantial step reductions, these results show that ExpMem offers a favorable balance between task success, interaction demand, and inference expenditure on ALFWorld.

B Experimental Settings

We evaluate ExpMem with Qwen-3.7-Max and MiniMax-M3 through AgentBoard using fixed task lists. ScienceWorld runs in easy mode. Later rounds retry only unsolved tasks.

Setting	ALFWorld	ScienceWorld
Evaluation tasks	134	150 (30 types $\times$ 5)
Maximum time steps	30	60
Maximum attempts per task	3	3
Recent history (time steps)	2	2
Memory retrieval top-k	3	3

Table 4: Evaluation settings.

Memory and inputs. Each method starts with an independent empty memory store. ExpMem retains memory across attempts and allows reuse across tasks when the recorded conditions apply.

Generation settings. The expectation generator and Action Executor use the same backbone.

C ALFWorld Implementation and Prompts

C.1 State and Memory Representation

The current state records the agent's position, held object, reachable locations, and visible objects. Below is the state before step 4; only `reachable_locations` is shortened.

```
{
  "current_position": {"location_name": "cabinet 2", "status": "open"},
  "item_in_hand": {"item_name": null, "status": null},
  "reachable_locations": ["cabinet 1", "cabinet 3", "coffeemachine 1", "microwave
     1"],
  "visible_item_in_current_location": ["cup 2", "vase 1"]
}
```

The Planner returns `reason`, `expectation`, and `next_state`. The expectation annotation and the nested `next_state` JSON jointly express the intended next state. The Executor receives this prediction and reason, with the current state, task goal, and available actions.

Writing and retrieval. Code rules populate the four-part memory template from state, feedback, and available-action evidence. New mismatch records are written or merged at the end of the step; later successful evidence can update an existing record. Each record retains its source steps and update events. Retrieval filters records by reliability and recorded conditions, then ranks them by state matching, task relevance, and reliability. Retrieved records enter the Planner's input; the Executor does not retrieve memory independently.

C.2 EXPECTATION GENERATOR PROMPT

Key instructions in the ALFWorld prompt templates are highlighted in red.

expectation generator

You are the expectation generator for ALFWorld. Given the task goal and current state, predict an expected next state that advances the task or establishes a prerequisite. The expected next state must be reachable by one available action.

Current input

Task goal: {task_goal}
Current state JSON: {current_state}
<available_action> {available_actions} </available_action>
Recent action-observation history (up to two steps): {action_history}
Retrieved memory: {memory}

Memory guidance

Use retrieved memory before predicting the expected next state. Consider its recorded conditions and evidence. Give priority to the current state and available actions when they conflict with memory. Avoid repeating a recorded unsuccessful attempt under the same conditions.

Next-state prediction

Use only the following fields in `next_state`:

1. `current_position.location_name`: A destination in `reachable_locations`.
2. `current_position.status`: "open" or "closed" for the current container.
3. `item_in_hand.item_name`: The object to take, or `null` after placing the held object. After placement, set `item_name` and `status` to `null`.
4. `item_in_hand.status`: "heated", "cooled", "cleaned", "sliced", or "lighted", when the corresponding action is currently available.

If heat/cool/clean is available and needed for the task, predict the corresponding `item_in_hand.status`. This also applies when the object is inside the tool and `item_name` is `null`. For a look-at-in-light task, an available lamp-use action can support `item_in_hand.status="lighted"`. Do not predict taking the lamp to represent using it.

Use `current_position.status` only for container access. Opening a microwave or fridge is a prerequisite, not a heat or cool action. Predict a state that is not already achieved. If the current subgoal is complete, predict the next intended state needed for the task.

expectation annotation

Use the following fields:

- `type`: search, move, open, take, put, transform, or use.
- `target`: the object type, object instance, or location.
- `expected`: find, access, hold, place, change_status, light, or finish.

For search, set `type="search"`, target to the object type, and `expected="find"`. The `next_state` still describes one movement or container-access step. For a prerequisite, use move/open with `expected="access"`. Use take/hold, put/place, transform/change_status, or use/light for the corresponding expectation.

Output format

Return one valid JSON object with three top-level keys: `reason`, `expectation`, and `next_state`. Preserve the nested structure of the current state in `next_state` and include the fields needed to express the expected next state.

Example, when cup 2 is visible and can be taken:

```
{
  "reason": "Take cup 2 before heating it.",
  "expectation": {"type": "take", "target": "cup 2", "expected": "hold"},
  "next_state": {"item_in_hand": {"item_name": "cup 2"}}
}
```

Do not output an action. Do not output text outside the JSON.

C.3 Action Executor Prompt

Action Executor

You are the Action Executor for ALFWorld. Select an available action that moves the current state toward the expected next state.

Current input

Task goal: {task_goal}
Current state JSON: {current_state}
<available_action> {available_actions} </available_action>
expected next state: {predicted_next_state}
expectation generator's reason: {prediction_rationale}

Action selection

Use the current state, available actions, and expected next state to select one action. Copy the action exactly from <available_action>.

Output format

Return only a valid JSON object with one key:

```
{"action": "<copy exactly one available action>"}
```

D Execution Trajectory and Memory Records

Figure 5 shows all 12 steps of *put a hot cup in cabinet*. Each step includes the current state, expected next state, executed action, and environmental feedback. Subgoals annotate the expected next state; only reachable_locations is abbreviated.

The unsuccessful cup searches at steps 0–1 produce and extend M1. At step 6, a heating prediction at coffeemachine 1 has no supporting legal heat action, producing M2. M2 is retrieved before planning at step 7; successful heating with the microwave at step 9 then updates this record. The task ends when the heated cup is placed in the cabinet at step 11. Figure 6 shows the corresponding four-part memories.

This example comes from a task-local configuration with up to 50 actions per attempt and retrieval top-$k = 50$. The main evaluation follows the separate settings in Section B.

ALFWorld execution trajectory

Task: put a hot cup in cabinet. Steps 0–1

Step 0 **Write M1** No cup found.

Current state JSON


```
{
  "current_position": {
    "location_name":
      "middle of a room. Looking quickly
    around you",
    "status": null
  },
  "item_in_hand": {
    "item_name": null,
    "status": null
  },
  "reachable_locations": [
    "cabinet 1", "cabinet 2",
    "coffeemachine 1", "microwave 1"
  ],
  "visible_item_in_current_location": [
    "cabinet 6", "cabinet 5", "cabinet 4",
    "cabinet 3", "cabinet 2", "cabinet 1",
    "coffeemachine 1", "countertop 3",
    "countertop 2", "countertop 1",
    "drawer 3", "drawer 2", "drawer 1",
    "fridge 1", "garbagecan 1",
    "microwave 1", "shelf 3", "shelf 2",
    "shelf 1", "sinkbasin 1",
    "stoveburner 4", "stoveburner 3",
    "stoveburner 2", "stoveburner 1",
    "toaster 1"
  ]
}
```


expected next state JSON

Subgoal: search cup


```
{
  "current_position": {
    "location_name":
      "cabinet 1"
  }
  "item_in_hand": {
     "cup"
  }
}
```


Action

`go to cabinet 1`

Environmental feedback

On the cabinet 1, you see a saltshaker 1.

Step 1 **Extend M1** No cup found.

Current state JSON


```
{
  "current_position": {
    "location_name": "cabinet 1",
    "status": null
  },
  "item_in_hand": {
    "item_name": null,
    "status": null
  },
  "reachable_locations": [
    "cabinet 2", "cabinet 3",
    "countertop 1", "microwave 1"
  ],
  "visible_item_in_current_location": [
    "saltshaker 1"
  ]
}
```


expected next state JSON

Subgoal: search cup


```
{
  "current_position": {
    "location_name":
      "countertop 1"
  }
  "item_in_hand": {
     "cup"
  }
}
```


Action

`go to countertop 1`

Environmental feedback

On the countertop 1, you see a cellphone 2, a egg 2, and a lettuce 1.

Figure 5: ALFWorld execution trajectory, steps 0–1. Current and predicted next states retain the recorded JSON structure and values, with reachable locations abbreviated. Subgoals annotate the intended-next-state panels; actions and feedback are reproduced from the trace.

ALFWorld execution trajectory

Task: put a hot cup in cabinet. Steps 2–3

Step 2

Current state JSON


```
{
  "current_position": {
    "location_name": "countertop 1",
    "status": null
  },
  "item_in_hand": {
    "item_name": null,
    "status": null
  },
  "reachable_locations": [
    "cabinet 1", "cabinet 2", "cabinet 3",
    "microwave 1"
  ],
  "visible_item_in_current_location": [
    "cellphone 2", "egg 2", "lettuce 1"
  ]
}
```


expected next state JSON

Subgoal: search cup


```
{
  "current_position": {
    "location_name":
      "cabinet 2"
  }
}
```


Action

```
go to cabinet 2
```

Environmental feedback

The cabinet 2 is closed.

Step 3

Current state JSON


```
{
  "current_position": {
    "location_name": "cabinet 2",
    "status": "closed"
  },
  "item_in_hand": {
    "item_name": null,
    "status": null
  },
  "reachable_locations": [
    "cabinet 1", "cabinet 3",
    "coffeemachine 1", "microwave 1"
  ],
  "visible_item_in_current_location": [
    "cellphone 2", "egg 2", "lettuce 1"
  ]
}
```


expected next state JSON

Subgoal: open cabinet 2


```
{
  "current_position": {
    "status": "open"
  }
}
```


Action

```
open cabinet 2
```

Environmental feedback

You open the cabinet 2. The cabinet 2 is open. In it, you see a cup 2, and a vase 1.

Figure 5: ALFWorld execution trajectory, steps 2–3 (continued). Reachable locations are abbreviated; other displayed JSON values are unchanged.

ALFWorld execution trajectory

Task: put a hot cup in cabinet. Steps 4–5

Step 4

Current state JSON


```
{
  "current_position": {
    "location_name": "cabinet 2",
    "status": "open"
  },
  "item_in_hand": {
    "item_name": null,
    "status": null
  },
  "reachable_locations": [
    "cabinet 1", "cabinet 3",
    "coffeemachine 1", "microwave 1"
  ],
  "visible_item_in_current_location": [
    "cup 2", "vase 1"
  ]
}
```


expected next state JSON

Subgoal: take cup 2


```
{
  "item_in_hand": {
    "item_name": "cup 2"
  }
}
```


Action

```
take cup 2 from cabinet 2
```

Environmental feedback

You pick up the cup 2 from the cabinet 2.

Step 5

Current state JSON


```
{
  "current_position": {
    "location_name": "cabinet 2",
    "status": "open"
  },
  "item_in_hand": {
    "item_name": "cup 2",
    "status": null
  },
  "reachable_locations": [
    "cabinet 1", "cabinet 3",
    "coffeemachine 1", "microwave 1"
  ],
  "visible_item_in_current_location": [
    "vase 1"
  ]
}
```


expected next state JSON

Subgoal: move to coffeemachine 1


```
{
  "current_position": {
    "location_name":
      "coffeemachine 1"
  }
}
```


Action

```
go to coffeemachine 1
```

Environmental feedback

On the coffeemachine 1, you see nothing.

Figure 5: ALFWorld execution trajectory, steps 4–5 (continued). Reachable locations are abbreviated; other displayed JSON values are unchanged.

ALFWorld execution trajectory

Task: put a hot cup in cabinet. Steps 6–7

Step 6 **Write M2** No legal heat action.

Current state JSON


```
{
  "current_position": {
    "location_name": "coffeemachine 1",
    "status": null
  },
  "item_in_hand": {
    "item_name": "cup 2",
    "status": null
  },
  "reachable_locations": [
    "cabinet 1", "cabinet 2", "cabinet 3",
    "microwave 1"
  ],
  "visible_item_in_current_location": []
}
```


expected next state JSON

Subgoal: heat cup 2


```
{
  "item_in_hand": {
    "status": "heated"
  }
}
```


Action

```
heat cup 2
```

Environmental feedback

Nothing happen.

Step 7 **Retrieve M2** Before planning.

Current state JSON


```
{
  "current_position": {
    "location_name": "coffeemachine 1",
    "status": null
  },
  "item_in_hand": {
    "item_name": "cup 2",
    "status": null
  },
  "reachable_locations": [
    "cabinet 1", "cabinet 2", "cabinet 3",
    "microwave 1"
  ],
  "visible_item_in_current_location": []
}
```


expected next state JSON

Subgoal: move to microwave 1


```
{
  "current_position": {
    "location_name":
      "microwave 1",
    "status": null
  }
}
```


Action

```
go to microwave 1
```

Environmental feedback

The microwave 1 is closed.

Figure 5: ALFWorld execution trajectory, steps 6–7 (continued). Reachable locations are abbreviated; other displayed JSON values are unchanged.

ALFWorld execution trajectory

Task: put a hot cup in cabinet. Steps 8–9

Step 8

Current state JSON


```
{
  "current_position": {
    "location_name": "microwave 1",
    "status": "closed"
  },
  "item_in_hand": {
    "item_name": "cup 2",
    "status": null
  },
  "reachable_locations": [
    "cabinet 1", "cabinet 2", "cabinet 3",
    "coffeemachine 1"
  ],
  "visible_item_in_current_location": []
}
```


expected next state JSON

Subgoal: open microwave 1


```
{
  "current_position": {
    "status": "open"
  }
}
```


Action

`open microwave 1`

Environmental feedback

You open the microwave 1. The microwave 1 is open. In it, you see nothing.

Step 9 Successful heating.

Current state JSON


```
{
  "current_position": {
    "location_name": "microwave 1",
    "status": "open"
  },
  "item_in_hand": {
    "item_name": "cup 2",
    "status": null
  },
  "reachable_locations": [
    "cabinet 1", "cabinet 2", "cabinet 3",
    "coffeemachine 1"
  ],
  "visible_item_in_current_location": []
}
```


expected next state JSON

Subgoal: heat cup 2


```
{
  "item_in_hand": {
    "status": "heated"
  }
}
```


Action

`heat cup 2 with microwave 1`

Environmental feedback

You heat the cup 2 using the microwave 1.

Figure 5: ALFWorld execution trajectory, steps 8–9 (continued). Reachable locations are abbreviated; other displayed JSON values are unchanged.

ALFWorld execution trajectory

Task: put a hot cup in cabinet. Steps 10–11

Step 10

Current state JSON


```
{
  "current_position": {
    "location_name": "microwave 1",
    "status": "open"
  },
  "item_in_hand": {
    "item_name": "cup 2",
    "status": "heated"
  },
  "reachable_locations": [
    "cabinet 1", "cabinet 2", "cabinet 3",
    "coffeemachine 1"
  ],
  "visible_item_in_current_location": []
}
```


expected next state JSON

Subgoal: move to cabinet 1


```
{
  "current_position": {
    "location_name":
      "cabinet 1"
  }
}
```


Action

`go to cabinet 1`

Environmental feedback

On the cabinet 1, you see a saltshaker 1.

Step 11 **Task complete**

Current state JSON


```
{
  "current_position": {
    "location_name": "cabinet 1",
    "status": null
  },
  "item_in_hand": {
    "item_name": "cup 2",
    "status": "heated"
  },
  "reachable_locations": [
    "cabinet 2", "cabinet 3",
    "coffeemachine 1", "microwave 1"
  ],
  "visible_item_in_current_location": [
    "saltshaker 1"
  ]
}
```


expected next state JSON

Subgoal: place cup 2


```
{
  "item_in_hand": {
    "item_name": null,
    "status": null
  }
}
```


Action

`put cup 2 in/on cabinet 1`

Environmental feedback

You put the cup 2 in/on the cabinet 1.

Figure 5: ALFWorld execution trajectory, steps 10–11 (continued). Reachable locations are abbreviated; other displayed JSON values are unchanged.

Memory records from the execution trajectory

Four components organize the recorded evidence for display.

M1 Cup search after step 1

Written at step 0; extended at step 1

Context

Task: put a hot cup in cabinet.
Search target: cup. **Locations:** cabinet 1 and countertop 1.

Expectation

Subgoal: search cup

Step 0 next_state

```
{
  "current_position": {
    "location_name":
      "cabinet 1"
  }
}
```

Step 1 next_state

```
{
  "current_position": {
    "location_name":
      "countertop 1"
  }
}
```

Evidence

Step 0 action: `go to cabinet 1`
Feedback: On the cabinet 1, you see a saltshaker 1.

Step 1 action: `go to countertop 1`
Feedback: On the countertop 1, you see a cellphone 2, a egg 2, and a lettuce 1.
No cup was observed at either location.

Constraint

Avoid repeating the cup search at cabinet 1 or countertop 1 while their observed contents remain unchanged.

M2 One-step heating at step 6

Written at step 6; retrieved before planning at step 7

Context

Location: coffeemachine 1. **Held object:** cup 2.
Task requirement: heat the cup before placing it in a cabinet.

Expectation

Subgoal: heat cup 2

Step 6 next_state

```
{
  "item_in_hand": {
    "status": "heated"
  }
}
```

Evidence

Before execution: **no legal heat action was available.**

Step 6 action: `examine coffeemachine 1`
Feedback: On the coffeemachine 1, you see nothing.
The recorded `item_in_hand.status` is `null` before and after this step.

Constraint

Do not repeat this one-step heating prediction while the recorded conditions and available actions remain unchanged.

Figure 6: Four-part presentation of the memories in the trajectory. M1 combines the cup-search evidence from steps 0 and 1. M2 records the heating prediction and the absence of a legal heat action at step 6; its later update adds successful heating evidence from step 9. Subgoals annotate expectation; JSON blocks reproduce the recorded predictions.