EffectMem: Effectiveness-Driven Memory Reuse for Long-Horizon LLM Agents

Jingjing Yin[1,*], **Lingbo Wang**[1,*], **Dongyan Cao**[2,†], **Yu Hong**[1,†]
[1]School of Computer Science and Technology, Soochow University, Suzhou, China
[2]Suzhou Research Institute, Harbin Institute of Technology, Suzhou, China
{jjyinniyjj, vllbc66, tianxianer}@gmail.com
dongyanhit@126.com

Abstract

Recent advances in Large Language Models (LLMs) have stimulated increasing interest in LLM-based agents for long-horizon interactive decision-making. As a lightweight alternative to parameter-level optimization, many studies enhance LLM agents through retrievable experience memory. However, existing retrieval strategies primarily optimize for the relevance of historical memories, leaving the applicability conditions and verifiable execution effects of retrieved trajectories insufficiently captured. To mitigate this executability gap between relevance-driven retrieval and executable reuse, we propose Effectiveness-Driven Memory (EffectMem), a framework that reframes trajectory memory as executable guidance whose applicability and effects are checked during interaction. During memory construction, EffectMem structures successful trajectories into positive executable contracts and localizable failures into negative experience guards. At inference, EffectMem retrieves candidate contracts, verifies their applicability to the current state, executes aligned chunks under guard constraints, and invokes discrepancy-guided correction when observed effects deviate from expected postconditions. Experiments on ALFWorld, ScienceWorld, and WebShop demonstrate that EffectMem outperforms competitive baselines and improves the effectiveness of memory-augmented LLM agents.

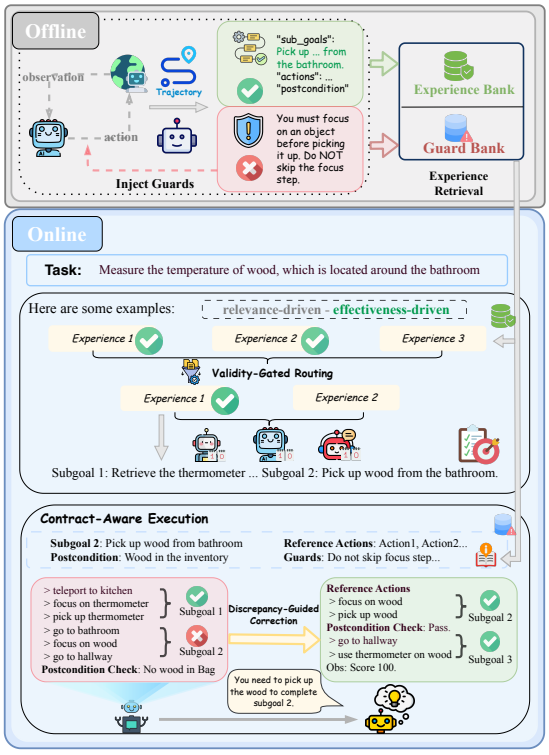


Figure 1: An illustrative example of EffectMem in ScienceWorld. EffectMem constructs executable contracts and negative guards offline, selects an applicable contract via validity-gated routing, and executes aligned chunks with postcondition-based correction.

1 Introduction

The remarkable progress of Large Language Models (LLMs) has stimulated increasing interest in LLM-based autonomous agents (Wang et al., 2024a) for complex, long-horizon interactive tasks (Yao et al., 2023b; Ahn et al., 2022; Yao et al., 2023a). These agents derive state-conditioned actions by grounding task objectives in dynamically evolving observations, which requires them to maintain procedural consistency across multiple state transitions. To support long-horizon execution without parameter-level adaptation, recent memory-augmented agents externalize reflections (Shinn et al., 2023), trajectories (Zhao et al., 2024a), workflows (Wang et al., 2024b), or skills (Qiu et al., 2026) as retrievable procedural memory for reuse. Among these memory forms, trajectory-level memory is especially valuable because it preserves procedural evidence across interaction states and feedback, making retrieval conditions and execution

[*]These authors contributed equally to this work.
[†]Corresponding authors.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 18175–18195
October 24-29, 2026 ©2026 Association for Computational Linguistics

effects crucial for reliable reuse.

Despite the value of trajectory-level memory, existing retrieval strategies rely heavily on the relevance of historical memories (namely relevance-driven strategies) (Zhao et al., 2024a), while making limited use of conditionally-interpretable and executively-effective memories. In this setting, a historical trajectory is likely to be retrieved when its task description or procedural summary appears semantically relevant to the current task. However, semantic relevance provides limited evidence about whether the trajectory remains executable under the current state. A retrieved memory may rely on objects, constraints, or intermediate states that are missing, have changed, or no longer fit the current task (Ahn et al., 2022). As a result, a semantically relevant memory may still induce invalid or non-executable guidance.

This problem is exacerbated by coarse-grained trajectory representation. Successful episodes are often reused as holistic demonstrations, which obscure the applicability conditions and segment-level execution effects required for executable reuse. Failed episodes are likewise underutilized, as they are discarded or stored as generic reflections rather than operationalized as constraints against recurring failure modes (Zhao et al., 2024a). Consequently, existing relevance-driven strategies can retrieve historical memories, but they do not sufficiently determine when those memories are applicable, how they should be executed, or whether their expected effects have been achieved.

To address this gap, we propose **EffectMem**, an **Effect**iveness-Driven **Mem**ory framework that recasts historical trajectories from relevance-based retrieval contexts into executable guidance grounded in applicability conditions and verifiable execution effects. As illustrated in Figure 1, EffectMem derives two memory banks from offline trajectories. Successful trajectories are abstracted into positive executable contracts, each comprising a retrieval summary, a global applicability condition, and a sequence of executable chunks. Each chunk specifies a subgoal, a reference action sequence, and an expected postcondition, rendering memory condition-aware before execution and effect-verifiable after execution. Localizable failures are iteratively distilled into negative experience guards, which provide guard rules for rerunning failed tasks and suppressing recurrent failure patterns.

During inference, EffectMem reuses memory through validity-gated routing and contract-aware execution. Validity-gated routing first retrieves candidate contracts according to the relevance of historical memories and then verifies whether their global applicability conditions are satisfied under the current task. It further employs multi-agent assessment to identify a contract that is not only relevant but also operationally suitable for the current task. Contract-aware execution executes the aligned chunks under activated negative guards, uses a monitor agent to determine subgoal completion and verify expected postconditions, and invokes discrepancy-guided correction when the completed subgoal fails to satisfy its expected postcondition. In this way, EffectMem shifts memory reuse from relevance-driven retrieval to effectiveness-driven execution, where a historical memory must be applicable before use and verifiable after use.

By grounding trajectory reuse in state applicability and execution effectiveness, EffectMem provides a validity-aware alternative to retrieval. Our contributions are summarized as follows:

- We identify a fundamental executability gap in trajectory-level memory reuse, revealing that semantically relevant trajectories may still lack the state conditions and verifiable effects required for valid experience reuse.
- We propose EffectMem, an effectiveness-driven memory framework that structures successful trajectories into positive executable contracts and localizable failures into negative experience guards, and operationalizes them through validity-gated routing and contract-aware execution at inference time.
- We conduct extensive experiments across multiple interactive benchmarks and LLM backbones, together with ablation, memory-scale, and top-k analyses, demonstrating that EffectMem consistently strengthens memory-augmented long-horizon agent execution.

2 Related Work

2.1 Experience Memory for LLM Agents

LLM-based agents solve interactive tasks by interleaving language reasoning with environment actions (Yao et al., 2023b). To enable cross-episode experience reuse, prior work equips agents with external memories distilled from interaction trajectories. Some methods derive textual or instructional memories from offline experience, including reusable insights (Zhao et al., 2024a; Fu et al.,

2024a), state-conditioned guidelines (Fu et al., 2024b), and instruction manuals (Chen et al., 2024). Beyond textual memories, recent studies explore structured memory forms (Li et al., 2024), including multi-grained grounded memories (Yang et al., 2026), hierarchical working memories (Hu et al., 2024; Sun and Zeng, 2025), agentic memory networks (Xu et al., 2025; Liang et al., 2025), procedural repositories (Fang et al., 2026), and self-refined trajectory expertise (Qiu et al., 2026; Cui et al., 2026). Although these studies show the value of externalized experience, EffectMem moves beyond contextual guidance by representing memories as executable contracts and guards with applicability checks, effect verification, and failure constraints.

2.2 Hierarchical Planning with Retrieved Experience

Long-horizon tasks require agents to preserve global task structure while adapting to local observations. Existing methods address this requirement through reasoning-acting agents (Yao et al., 2023b), hierarchical planning (Zhao et al., 2024b) with milestone guidance (Li et al., 2025), and explicit planning-execution separation (Erdogan et al., 2025). Other approaches maintain long-horizon dependencies by compressing execution histories around subgoals (Hu et al., 2024), constructing tree-based agent control structures (Choi et al., 2026), or adding memory-enhanced planning modules for embodied tasks (Glocker et al., 2025). Retrieval-enhanced planning (Sun et al., 2024) reuses prior experience for stepwise decision making (Zhou et al., 2024) and for task decomposition and action generation in web agents (Kim et al., 2024a). However, existing retrieval strategies (Yao et al., 2024) rely on the relevance of historical memories, whereas EffectMem verifies contract-level applicability to avoid state-inapplicable reuse.

2.3 Reflection and Failure-Aware Correction

Feedback-based methods improve agents by converting failures into corrective signals. Reflexion (Shinn et al., 2023; Zhang et al., 2024) stores verbal reflections from failed attempts, while Trial and Error (Song et al., 2024) uses failed trajectories to optimize future behavior. QuBE (Kim et al., 2024b) constructs question-based belief states to reduce reasoning derailment, and QueryAgent (Huang et al., 2024) performs self-correction using environment feedback. Recent memory safety work further stores detected failures as defensive lessons for future actions (Wei et al., 2025). Although failure feedback is useful, prior methods usually encode it as textual signals, whereas EffectMem turns localized failures into explicit reusable pre-execution guards and triggers correction when postcondition verification fails.

3 Approach

This section presents the three stages of EffectMem, as illustrated in Figure 2. Offline Executable Memory Construction derives two complementary memory banks from training trajectories: positive executable contracts from successful trajectories and negative experience guards from localizable failures (§3.1). Validity-Gated Routing retrieves candidate contracts according to the relevance of historical memories, verifies their contract-level applicability under the current state, and selects an aligned contract for execution (§3.2). Contract-Aware Execution executes the aligned chunks under negative-guard constraints, uses a monitor agent to judge subgoal completion and verify expected postconditions, and invokes discrepancy-guided correction when verification fails (§3.3).

3.1 Offline Executable Memory Construction

EffectMem collects trajectories by running a base agent π_{base} on training tasks and partitions them according to execution outcomes, as illustrated in Algorithm 1 (Appendix A). Successful trajectories are abstracted into positive executable contracts, as they provide procedural evidence of how a task can be executed from initial conditions to successful completion. For each successful trajectory, an LLM-based analyzer generates a retrieval summary S_i and decomposes the trajectory into subgoal-level executable chunks. The resulting contract is stored as $e_i^+ = \langle S_i, G_i, C_i \rangle$ in the positive memory bank $\mathcal{M}^+$, where S_i supports retrieval according to historical-memory relevance, G_i specifies the global applicability condition, and C_i stores the chunk sequence for contract-aware execution.

The global applicability condition G_i is derived from the task instruction, initial observation, and trajectory-level prerequisites that determine whether the historical trajectory can be reused under a new task. Each chunk $c_{ij} = \langle d_{ij}, a_{ij}, q_{ij} \rangle \in C_i$ contains a subgoal description d_{ij}, a reference action sequence a_{ij}, and an expected postcondition q_{ij}. The analyzer preserves the executed action span as a_{ij} and extracts q_{ij} from the corre-

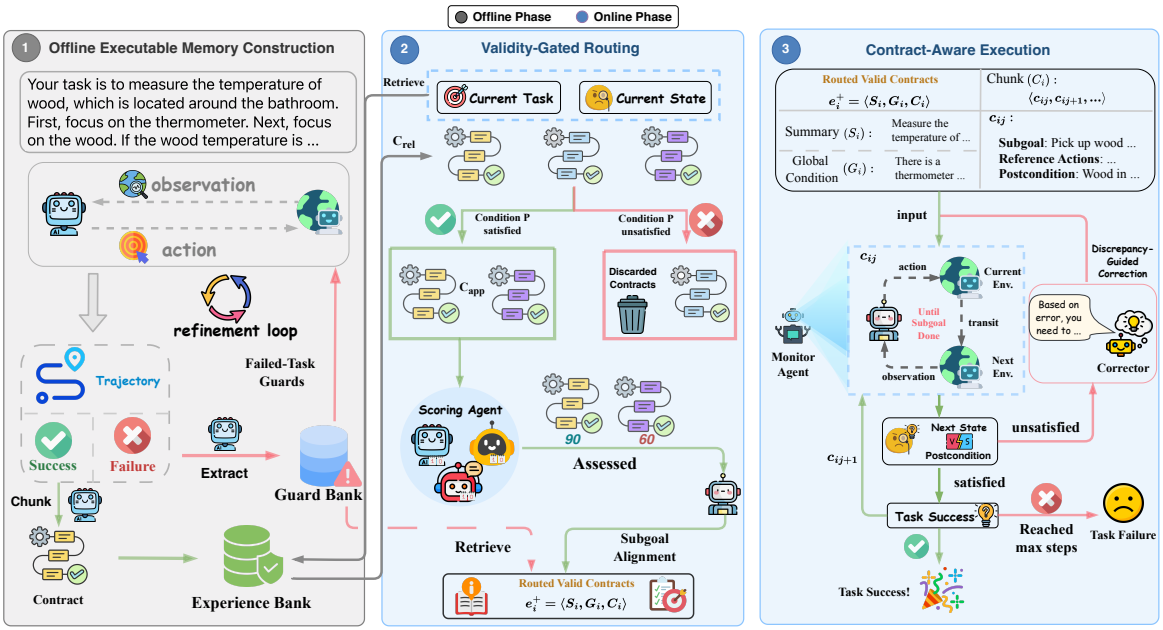


Figure 2: Overview of EffectMem. EffectMem builds executable contracts and experience guards from successful and failed trajectories offline, then reuses them through validity-gated routing and contract-aware execution. Aligned chunks are executed under guard constraints, verified by postconditions against the current state, and corrected when discrepancies arise. Green and red arrows indicate success and failure/correction paths.

sponding state transition and environment feedback. This representation renders each chunk executable through reference actions and verifiable through its expected postcondition after execution.

For failed trajectories, EffectMem focuses on localizable failures whose causes can be associated with the task context and execution feedback. Such failures are distilled into negative experience guards rather than retained as generic textual feedback. Given a failed task τ and its failed trajectory H_τ, an LLM-based analyzer extracts a guard from the failure feedback. We represent each guard as $g_i^- = \langle T_i^-, R_i^- \rangle$, where T_i^- denotes the failed task description and R_i^- contains guard rules for avoiding the localized failure. The guard is stored in the negative guard bank $\mathcal{M}^-$ and is used to rerun the failed task during memory refinement.

To validate and refine negative experience guards, EffectMem reruns the corresponding failed task under the extracted defensive guard. If the rerun succeeds, the resulting trajectory is analyzed and promoted to $\mathcal{M}^+$ as a positive executable contract. If the rerun still fails, EffectMem analyzes the new failed trajectory, extracts an additional defensive guard, and reruns the same failed task again. This guard extraction and rerun process is repeated until the task succeeds or the guard-rerun budget B is exhausted. When the budget is exhausted, the accumulated guards remain in $\mathcal{M}^-$ as reusable defensive constraints for later execution.

3.2 Validity-Gated Routing

Given a new task t, current state S_t, and the positive memory bank $\mathcal{M}^+$, EffectMem selects an executable contract through retrieval, applicability verification, and contract-level assessment. As illustrated in Algorithm 2, the retriever first constructs a candidate set C_{rel} according to the relevance between the current task and the retrieval summary S_i of each historical contract. This step preserves the utility of retrieval while treating relevance as a necessary but insufficient condition for reuse.

EffectMem then verifies whether the global applicability condition G_i of each candidate contract $e_i^+ = \langle S_i, G_i, C_i \rangle$ is satisfied under the current state S_t. Contracts that fail this check are filtered out, and the remaining contracts form the applicable set C_{app}. This gate prevents contracts from being reused solely on the basis of semantic relevance, without satisfying global prerequisites.

To select the contract, EffectMem performs multi-agent assessment over C_{app}. Rather than relying on a relevance score, the assessment decomposes contract utility into three complementary dimensions: task-strategy consistency, subgoal compatibility, and action feasibility. Task-strategy

consistency examines whether a contract follows a high-level procedure compatible with the task, subgoal compatibility evaluates whether its executable chunks cover intermediate objectives, and action feasibility assesses whether the reference actions are executable under the state and environment constraints. The scores from the corresponding assessment agents are normalized and aggregated into a contract-level utility score, and the highest-scoring contract is selected as the execution reference. This assessment further reduces the risk of selecting a contract that is applicable in a broad sense but strategically or operationally unsuitable for the current task.

After selection, EffectMem aligns the selected contract with the current task context and state. The alignment keeps chunks relevant to the current execution progress, removes irrelevant chunks, and instantiates entity-specific postconditions when necessary. The resulting aligned chunks are then passed to contract-aware execution. Together, retrieval, applicability verification, assessment, and alignment convert historical-memory relevance into executable selection grounded in current-state applicability and expected execution effects.

3.3 Contract-Aware Execution

After validity-gated routing selects and aligns a contract, EffectMem executes the aligned chunks instead of treating the retrieved trajectory as a demonstration to be replayed verbatim. Since these chunks are derived from the ordered sequence of a contract, they preserve the temporal and logical dependencies needed for coherent execution. For each aligned chunk $\tilde{c}_j = \langle d_j, a_j, q_j \rangle$, the subgoal description d_j specifies the execution objective, the reference action sequence a_j guides action generation, and the expected postcondition q_j supports effect verification, as illustrated in Algorithm 3.

Before executing aligned chunks, EffectMem retrieves task-relevant guards from the negative guard bank $\mathcal{M}^-$ according to the task description. These guards contain rules extracted from localized failures and are incorporated into the execution context as constraints. Thus, the selected positive contract provides a structured execution scaffold, while the retrieved negative guards help the executor avoid failure patterns observed in similar tasks.

After executing a chunk under these constraints, EffectMem invokes a monitor agent to determine whether the current subgoal has been completed. The monitor agent examines the executor's thinking history, executed actions, and environment feedback, and then verifies whether the expected postcondition q_j is satisfied once the subgoal is completed. If q_j is satisfied, execution proceeds to the next aligned chunk. Otherwise, the unmet postcondition is recorded as a discrepancy between the expected effect and the observed outcome, triggering discrepancy-guided correction for localized recovery. This mechanism ensures that reused memory is not assumed valid after retrieval, but is continuously monitored and verified through expected effects throughout execution.

Unlike negative guards, which prevent known failure patterns before execution, discrepancy-guided correction provides recovery after an observed outcome deviates from the expected postcondition. When such a discrepancy occurs, EffectMem invokes an LLM to generate corrective guidance based on the unmet postcondition, the executed chunk, the execution history, and the latest environment feedback. The corrective instruction is appended only to the current execution context and is not written back to $\mathcal{M}^+$ or $\mathcal{M}^-$, keeping both memory banks fixed during evaluation. Concrete examples of discrepancy-guided correction, guard extraction, and monitor-based verification are provided in Appendix F.

4 Experimentation

We evaluate EffectMem through three complementary empirical analyses. First, we assess its overall effectiveness against representative baselines across three long-horizon interactive environments. Second, we perform ablations to disentangle the contribution of validity-gated routing and contract-aware execution. Third, we analyze memory reuse under memory-bank perturbations to examine whether the gains arise from applicability-grounded and effect-verifiable reuse rather than relevance-driven retrieval alone.

4.1 Experimental Setup

Environments. We evaluate EffectMem on three interactive environments that require long-horizon decision making under dynamically changing observations. ALFWorld (Shridhar et al., 2021) requires agents to complete household tasks by navigating rooms and manipulating objects. ScienceWorld (Wang et al., 2022) evaluates procedural scientific reasoning that requires ordered actions and intermediate state changes. WebShop (Yao

Method	Features			ALFWorld	ScienceWorld	WebShop	
	CE-Dist	*CS-Route*	*EV-Reuse*	Success Rate (SR) ↑	SR ↑	Reward ↑	SR ↑
ReAct	✗	✗	✗	80.6%	43.0%	58.6	37.0%
ExpeL	✓	✓	✗	81.3%	57.0%	62.2	42.0%
AutoGuide	✓	✓	✗	83.6%	-	73.3	47.0%
QuBE	✗	✗	✗	84.3%	-	-	-
ExpeL + QuBE	✓	✓	✗	85.1%	65.0%	75.6	49.0%
AutoRefine	✓	✓	✗	-	70.4%	-	-
KnowMap	✗	✓	✗	-	<u>76.3%</u>	-	-
CFGM	✓	✓	✗	<u>91.0%</u>	74.0%	<u>85.0</u>	<u>57.0%</u>
EffectMem	✓	✓	✓	**93.8%**	**79.0%**	**85.8**	**73.1%**

Table 1: Main results on ALFWorld, ScienceWorld, and WebShop under the GPT-4-Turbo model. Dashes indicate unavailable results under the corresponding evaluation setting. CE-Dist, CS-Route, and EV-Reuse denote contrastive experience distillation, current-state memory routing, and effect-verifiable reuse, respectively. Best results are in bold and second-best results are underlined.

et al., 2023a) evaluates web-based search and purchasing decisions under attribute constraints. More environment details are provided in Appendix B.

Metrics. We report success rate (SR) as the main metric across all environments. SR measures task completion in ALFWorld, target-task completion in ScienceWorld, and attribute-satisfying purchase success in WebShop. For WebShop, we also report mean reward, which measures how closely the purchased item matches the required attributes.

Baselines. We compare EffectMem with ReAct (Yao et al., 2023b) and representative memory-based or feedback-based baselines, including ExpeL (Zhao et al., 2024a), AutoGuide (Fu et al., 2024b), QuBE (Kim et al., 2024b), ExpeL + QuBE, AutoRefine (Qiu et al., 2026), KnowMap (Fu and Bian, 2025), and CFGM (Yang et al., 2026), where reported results are available. The main comparison uses GPT-4-Turbo, and additional model results are reported in Table 3.

Implementation Details. Unless otherwise specified, EffectMem retrieves the top 5 candidate contracts before applying validity-gated routing by default. Details of model and implementation settings are given in Appendix C.

4.2 Main Results

Table 1 compares EffectMem with representative baselines on ALFWorld, ScienceWorld, and WebShop. We analyze the results from two perspectives, with emphasis on overall effectiveness and performance across task environments.

Overall Effectiveness. EffectMem consistently achieves the strongest performance across all reported environments and metrics. Compared with ReAct, EffectMem enhances ALFWorld SR by 13.2 points, ScienceWorld SR by 36.0 points, WebShop SR by 36.1 points, and WebShop reward by 27.2. These gains indicate that EffectMem's advantage is not confined to a particular environment, but yields gains in embodied interaction, scientific reasoning, and web-based decision making. They also indicate that effective trajectory reuse requires more than relevance-based retrieval: retrieved contracts must be applicable to the current state and their execution effects must remain verifiable after alignment and execution.

Performance across Task Environments. EffectMem also outperforms existing memory-augmented methods. On ALFWorld, it improves over CFGM by 2.8 SR points. On WebShop, it raises the best reward from 85.0 to 85.8 and increases SR over CFGM from 57.0 to 73.1. These results suggest that applicability checking and effect verification reduce the risk of following memories that are relevant but not executable. This advantage is particularly evident on WebShop, where product-search trajectories can be superficially similar yet differ substantially in attribute constraints and purchase requirements.

4.3 Ablation Study

We conduct ablations on ScienceWorld with Qwen3-8B to disentangle the contribution of each mechanism. We examine validity-gated routing (VGR), which selects contracts according to current-state applicability, and contract-aware execution (CAE), which executes aligned chunks under guard constraints, uses a monitor agent to verify subgoal completion and expected postconditions,

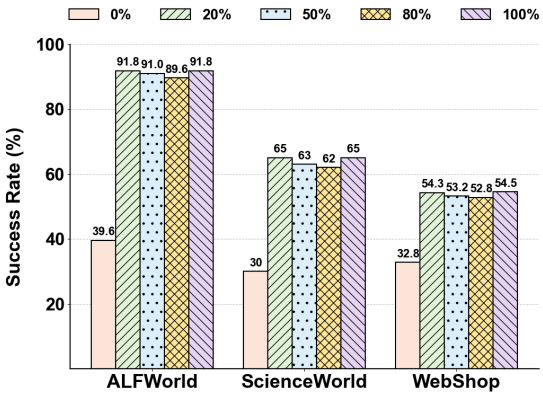
and invokes discrepancy-guided correction when verification fails. Here, CAE denotes the complete module, while (CAE)* removes discrepancy-guided correction. This design allows us to isolate the contribution of discrepancy-guided correction by comparing V&C* with V&C. For the CAE-only setting, the execution reference is taken from the highest-ranked contract under relevance-driven retrieval and then aligned to the current task without contract-level applicability filtering or multi-agent assessment. ScienceWorld is particularly suitable for this analysis because successful execution hinges on ordered subgoals, state-dependent applicability, and verifiable intermediate effects. Table 2 reports the official task score (Score), success rate (SR), and the average number of execution steps (Step) for each configuration.

Config	Score ↑	SR ↑	Δ	Step ↓	Δ
Baseline	33.8	15%	—	44.0	—
+*VGR*	71.4	51%	+36%	30.1	-13.9
+(CAE)*	70.2	57%	+42%	28.1	-15.9
+*V&C**	71.4	62%	+47%	26.3	-17.7
+*V&C*	**76.0**	**65%**	**+50%**	**25.9**	**-18.1**

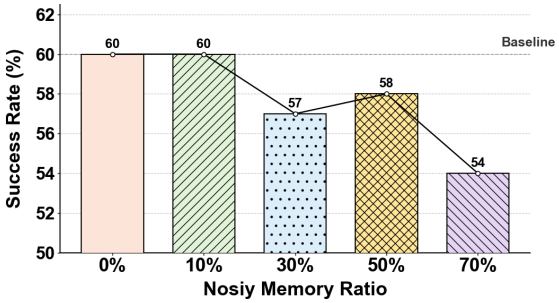
Table 2: Ablation study on ScienceWorld with Qwen3-8B. ↑ indicates higher is better and ↓ indicates lower is better. Deltas are absolute changes from the baseline. V&C* combines VGR with (CAE)*, and V&C combines VGR with the complete CAE.

As indicated by the arrows in Table 2, higher Score and SR indicate task success, while lower Step indicates more efficient execution. Adding VGR raises SR from 15% to 51% and shortens the execution length from 44.0 to 30.1 steps, indicating that applicability checking filters state-inapplicable memory reuse. Adding CAE without VGR improves SR to 57% and reduces average steps to 28.1, indicating that guard-constrained chunk execution and monitor-based postcondition verification are useful even without applicability-gated selection.

Combining VGR with (CAE)* raises SR to 62% and reduces average steps to 26.3. This suggests that contract-level applicability checking and execution-time verification address complementary failure modes. VGR enhances which contract is selected by filtering for current-state applicability, while CAE enhances how aligned chunks are executed by enforcing guards and monitoring expected postconditions. The full V&C configuration performs best, reaching 76.0 score, 65% SR, and 25.9 average steps. The gain from V&C* to V&C further demonstrates the benefit of discrepancy-guided correction when observed execution outcomes fail to satisfy expected postconditions.



(a) Memory-bank size.



(b) Noisy-memory injection.

Figure 3: Robustness to memory-bank perturbations. Memory-bank size varies task-balanced subsets of the clean memory bank, while noisy-memory injection mixes clean memories with perturbed memories. The limited performance drop indicates that EffectMem relies on applicability checking and effect verification rather than relevance-driven retrieval alone.

4.4 Analysis

Effectiveness across Models. Table 3 demonstrates that EffectMem generalizes across models with different capability levels. With Qwen3-Max, EffectMem reaches 93.3% SR on ALFWorld, 82.0% SR on ScienceWorld, and 67.2% SR on WebShop, outperforming both ReAct and ExpeL. With Qwen3-8B, it improves over ExpeL by 41.0 points on ALFWorld, 20.0 points on ScienceWorld, and 14.9 points on WebShop SR. These results indicate that applicability-grounded and effect-verifiable reuse provides structured procedural guidance that complements stronger models while substantially assisting smaller models.

Memory-bank Size (Zhang et al., 2025). Figure 3a varies the fraction of the memory bank available at inference time. Removing memory

Method	ALFWorld	ScienceWorld	WebShop	
	Success Rate (SR) ↑	SR ↑	Reward ↑	SR ↑
Qwen3-Max				
ReAct	59.7%	29.0%	52.9	44.1%
ExpeL	90.3%	66.0%	60.0	53.7%
EffectMem	**93.3%**	**82.0%**	**76.3**	**67.2%**
Qwen3-8B				
ReAct	38.3%	15.0%	49.6	28.3%
ExpeL	50.8%	45.0%	55.2	39.6%
EffectMem	**91.8%**	**65.0%**	**71.0**	**54.5%**

Table 3: Additional results on ALFWorld, ScienceWorld, and WebShop under Qwen3-Max and Qwen3-8B models. EffectMem consistently outperforms ReAct and ExpeL across models.

substantially degrades performance in all three environments, showing that executable experience provides useful procedural guidance. However, performance saturates once 20% of the memory bank is available, and increasing the bank to 100% yields only marginal changes. This pattern suggests that EffectMem is relatively insensitive to memory-bank size variation. Although the amount of available memory varies, task success changes only mildly, indicating that validity-gated routing effectively filters inapplicable memories and reduces dependence on memory quantity.

Noisy-Memory Injection. To evaluate robustness against corrupted but semantically relevant memories, we construct noisy memory banks from the original clean memory bank. The example of noisy memory is shown in Appendix D. Specifically, the clean memory bank is built by sampling 20% of the memories from each task category, so that the resulting bank preserves the category distribution of the original memory bank. We then generate noisy memories by perturbing clean memories with three corruption types: globally inapplicable conditions, chunk-level action or postcondition mutations, and failure patterns disguised as pseudo-positive memories. The generated noisy memories are mixed with the clean bank to form noisy-memory ratios of 10%, 30%, 50%, and 70%, where the noisy-memory ratio denotes the proportion of perturbed memories in the mixed memory bank.

As shown in Figure 3b, EffectMem exhibits only mild degradation as the noisy-memory ratio increases and remains close to the clean-memory setting. This result is consistent with the memory-size analysis: what matters is not whether a trajectory is retrievable, but whether it is applicable and executable under the current state. Validity-gated routing filters state-inapplicable candidates before execution, while contract-aware execution uses guard constraints and postcondition verification to suppress the influence of semantically plausible but corrupted memories.

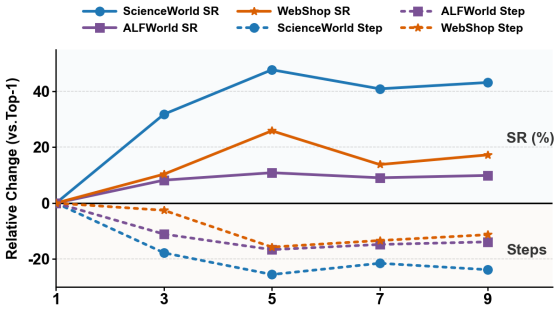


Figure 4: Top-k sensitivity analysis with Qwen3-8B. Relative to top-1 retrieval, $k = 5$ provides the best balance between candidate diversity and retrieval noise, improving SR while reducing average execution steps.

Top-k Sensitivity. Figure 4 reports changes relative to top-1 retrieval, where positive values indicate higher SR and negative values indicate fewer execution steps. Increasing k initially improves performance because VGR is exposed to a diverse candidate bank and selects one that better satisfies current-state applicability. The gains taper off when too many candidates are retrieved, as weakly applicable or invalid memories dilute the discriminative routing signal. We set $k = 5$ because it gives the largest SR gain while reducing execution length. Overall, these analyses suggest that EffectMem improves agent execution through applicability-grounded and effect-verifiable memory reuse rather than relevance-driven retrieval alone.

5 Conclusion

We introduced EffectMem, an effectiveness-driven memory framework for long-horizon LLM agents. EffectMem converts trajectories into executable contracts and failures into negative guards, enabling memory reuse to be checked, constrained during execution, and verified through effects. At inference, it verifies contract applicability, executes chunks under guard constraints, and triggers discrepancy-guided correction through postcondition verification during execution. Experiments on ALFWorld, ScienceWorld, and WebShop show gains over memory baselines across backends, confirming the importance of moving beyond relevance-driven retrieval toward applicability-grounded and effect-verifiable execution.

Limitations

Our work focuses on trajectory-level experience reuse for long-horizon LLM agents through applicability-aware and effect-verifiable memory augmentation. It still has some limitations. One limitation is that the quality of executable memory depends on the availability and diversity of collected trajectories. When training tasks are scarce or do not cover diverse task conditions, constructing reliable contracts and guards remains difficult. Another limitation is that EffectMem uses LLM-based analysis for contract construction, guard extraction, subgoal alignment, and monitor-based postcondition verification, which provides flexibility but may introduce additional cost and occasional judgment errors under ambiguous observations. Future work may explore more efficient trajectory collection, lightweight verification, and adaptive chunk selection to improve EffectMem under limited experience and more diverse tasks.

Acknowledgements

This research was supported by National Natural Science Foundation of China under No.62376182.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, and 26 others. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *Preprint*, arXiv:2204.01691.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. 2024. Automanual: Constructing instruction manuals by llm agents via interactive environmental learning. *Preprint*, arXiv:2405.16247.

Jae-Woo Choi, Hyungmin Kim, Hyobin Ong, Youngwoo Yoon, Minsu Jang, Dohyung Kim, and Jaehong Kim. 2026. Reactree: Hierarchical llm agent trees with control flow for long-horizon task planning. *Preprint*, arXiv:2511.02424.

Sijia Cui, Aiyao He, Shuai Xu, Hongming Zhang, Yanna Wang, Qingyang Zhang, Yajing Wang, and Bo Xu. 2026. Self-guided function calling in large language models via stepwise experience recall. *Preprint*, arXiv:2508.15214.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-act: Improving planning of agents for long-horizon tasks. *Preprint*, arXiv:2503.09572.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2026. Memp: Exploring agent procedural memory. *Preprint*, arXiv:2508.06433.

Dayuan Fu, Biqing Qi, Yihuai Gao, Che Jiang, Guanting Dong, and Bowen Zhou. 2024a. MSI-agent: Incorporating multi-scale insight into embodied agents for superior planning and decision-making. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 643–659, Miami, Florida, USA. Association for Computational Linguistics.

Kelin Fu and Kaigui Bian. 2025. Knowmap: Efficient knowledge-driven task adaptation for llms. *Preprint*, arXiv:2506.19527.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024b. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Preprint*, arXiv:2403.08978.

Marc Glocker, Peter Hönig, Matthias Hirschmanner, and Markus Vincze. 2025. Llm-empowered embodied agent for memory-augmented task planning in household robotics. *Preprint*, arXiv:2504.21716.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2024. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. *Preprint*, arXiv:2408.09559.

Xiang Huang, Sitao Cheng, Shanshan Huang, Jiayu Shen, Yong Xu, Chaoyun Zhang, and Yuzhong Qu. 2024. Queryagent: A reliable and efficient reasoning framework with environmental feedback-based self-correction. *Preprint*, arXiv:2403.11886.

Minsoo Kim, Victor Bursztyn, Eunyee Koh, Shunan Guo, and Seung-won Hwang. 2024a. RaDA: Retrieval-augmented web agent planning with LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13511–13525, Bangkok, Thailand. Association for Computational Linguistics.

Minsoo Kim, Jongyoon Kim, Jihyuk Kim, and Seung-won Hwang. 2024b. QuBE: Question-based belief enhancement for agentic LLM reasoning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21403–21423, Miami, Florida, USA. Association for Computational Linguistics.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. 2024. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. *Preprint*, arXiv:2408.03615.

Ziyue Li, Yuan Chang, Gaihong Yu, and Xiaoqiu Le. 2025. Hiplan: Hierarchical planning for llm-based agents with adaptive global-local guidance. *Preprint*, arXiv:2508.19076.

Xuechen Liang, Yangfan He, Yinghui Xia, Xinyuan Song, Jianhui Wang, Meiling Tao, Li Sun, Xinhang Yuan, Jiayi Su, Keqin Li, Jiaqi Chen, Jinsong Yang, Siyuan Chen, and Tianyu Shi. 2025. Self-evolving agents with reflective and memory-augmented abilities. *Preprint*, arXiv:2409.00872.

Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. 2026. Autorefine: From trajectories to reusable expertise for continual llm agent refinement. *Preprint*, arXiv:2601.22758.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. Alfworld: Aligning text and embodied environments for interactive learning. *Preprint*, arXiv:2010.03768.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for llm agents. *Preprint*, arXiv:2403.02502.

Chuanneng Sun, Songjun Huang, and Dario Pompili. 2024. Retrieval-augmented hierarchical in-context reinforcement learning and hindsight modular reflections for task planning with llms. *Preprint*, arXiv:2408.06520.

Haoran Sun and Shaoning Zeng. 2025. Hierarchical memory for high-efficiency long-term reasoning in llm agents. *Preprint*, arXiv:2507.22925.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024a. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6).

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11298, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Qianshan Wei, Tengchao Yang, Yaochen Wang, Xinfeng Li, Lijun Li, Zhenfei Yin, Yi Zhan, Thorsten Holz, Zhiqiang Lin, and XiaoFeng Wang. 2025. A-memguard: A proactive defense framework for llm-based agent memory. *Preprint*, arXiv:2510.02373.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *Preprint*, arXiv:2502.12110.

Wei Yang, Jinwei Xiao, Hongming Zhang, Qingyang Zhang, Yanna Wang, and Bo Xu. 2026. Coarse-to-fine grounded memory for llm agent planning. *Preprint*, arXiv:2508.15305.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2023a. Webshop: Towards scalable real-world web interaction with grounded language agents. *Preprint*, arXiv:2207.01206.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. 2024. Retroformer: Retrospective large language agents with policy gradient optimization. *Preprint*, arXiv:2308.02151.

Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. 2024. Agent-pro: Learning to evolve via policy-level reflection and optimization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5348–5375, Bangkok, Thailand. Association for Computational Linguistics.

Yunxiao Zhang, Guanming Xiong, Haochen Li, and Wen Zhao. 2025. Edge: Efficient data selection for llm agents via guideline effectiveness. *Preprint*, arXiv:2502.12494.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024a. Expel: Llm agents are experiential learners. *Preprint*, arXiv:2308.10144.

Qi Zhao, Haotian Fu, Chen Sun, and George Konidaris. 2024b. EPO: Hierarchical LLM agents with environment preference optimization. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 6401–6415, Miami, Florida, USA. Association for Computational Linguistics.

Ruiwen Zhou, Yingxuan Yang, Muning Wen, Ying Wen, Wenhao Wang, Chunling Xi, Guoqiang Xu, Yong Yu, and Weinan Zhang. 2024. Trad: Enhancing llm agents with step-wise thought retrieval and aligned decision. *Preprint*, arXiv:2403.06221.

A Algorithms

This section provides the three appendix algorithms referenced in the method section. Algorithm 1 describes offline executable memory construction. Algorithm 2 describes validity-gated routing. Algorithm 3 then executes the aligned chunks under guard constraints and postcondition verification.

B Environment Details

We evaluate EffectMem in three text-based interactive environments that require long-horizon action selection under changing observations. Table 4 summarizes the environment interfaces and evaluation settings used in our experiments.

ALFWorld. ALFWorld instantiates household tasks as text-based interaction episodes. The agent must infer the target object, identify required receptacles or appliances, satisfy preconditions, and complete the task within the step budget. We treat an episode as successful when the environment returns the terminal success signal. EffectMem is especially useful in ALFWorld because similar tasks often share procedural structure, but reusable experience still depends on task type, required facilities, destination receptacles, and the currently visible household state.

ScienceWorld. ScienceWorld requires agents to execute elementary science procedures in an ordered manner. Many tasks require intermediate operations whose effects must be achieved before later actions become valid. We report the official task score and success rate. This environment stresses the interaction between validity-gated routing and contract-aware execution: the selected contract must match the current scientific procedure, and each chunk must produce the expected state transition before the agent continues.

WebShop. WebShop presents natural-language purchase instructions and web-page observations. The agent can search for products, open search results, inspect product pages, select options such as size or color, and click buy now. We report both mean reward and success rate, where success requires buying an item satisfying the required attributes. WebShop contains many similar product-search trajectories, making it a strong test of whether memory retrieval checks applicability rather than following superficially similar examples.

Algorithm 1 Offline Executable Memory Construction

Require: Training tasks $\mathcal{D}_{\text{train}}$, base agent π_{base}, LLM analyzer $\mathcal{A}$, guard-rerun budget B
Ensure: Positive memory bank $\mathcal{M}^+$ and negative guard bank $\mathcal{M}^-$

$\mathcal{M}^+ \leftarrow \emptyset, \mathcal{M}^- \leftarrow \emptyset$
for all task $\tau \in \mathcal{D}_{\text{train}}$ **do**
  $H_\tau, y_\tau \leftarrow \textsc{RunEpisode}(\pi_{\text{base}}, \tau)$
  if $y_\tau =$ success **then**
    $S_\tau, G_\tau, C_\tau \leftarrow \textsc{AnalyzeSuccess}(\mathcal{A}, \tau, H_\tau)$
    $\mathcal{M}^+ \leftarrow \mathcal{M}^+ \cup \{\langle S_\tau, G_\tau, C_\tau \rangle\}$
  else
    $H_\tau^{(0)} \leftarrow H_\tau$
    for $b = 1, \ldots, B$ **do**
      $g_b^- = \langle T_b^-, R_b^- \rangle \leftarrow \textsc{ExtractGuard}(\mathcal{A}, \tau, H_\tau^{(b-1)})$
      $\mathcal{M}^- \leftarrow \mathcal{M}^- \cup \{g_b^-\}$
      $H_\tau^{(b)}, y_\tau^{(b)} \leftarrow \textsc{RunEpisode}(\pi_{\text{base}}, \tau, R_b^-)$
      if $y_\tau^{(b)} =$ success **then**
        $S_\tau^{(b)}, G_\tau^{(b)}, C_\tau^{(b)} \leftarrow \textsc{AnalyzeSuccess}(\mathcal{A}, \tau, H_\tau^{(b)})$
        $\mathcal{M}^+ \leftarrow \mathcal{M}^+ \cup \{\langle S_\tau^{(b)}, G_\tau^{(b)}, C_\tau^{(b)} \rangle\}$
        break
      end if
    end for
  end if
end for
return $\mathcal{M}^+, \mathcal{M}^-$

C Implementation Details

We adopted different experimental task selection strategies for different environments. In the case of ALFWorld we utilized the 134 solvable tasks that CFGM used, which comprising six task types in a virtual household environment: Pick (24), Clean (18), Heat (31), Cool (23), Examine (21), and Pick two (17). Similarly, for WebShop, we evaluated using the same 134 tasks used by CFGM. For ScienceWorld (Wang et al., 2022), we assess performance using 100 tasks sampled from the test set, which comprise diverse task types with environmental steps up to 50: use-thermometer (10), power-component (5), test-conductivity (10), find-animal (10), find-living-thing (10), find-non-living-thing (10), find-plant (10), grow-plant (8), chemistry-mix (7), chemistry-mix-paint-secondary-color (4), lifespan-longest-lived (3), identify-life-stages-1 (5), identify-life-stages-2 (4), and change-the-state-of-matter-of (4).

Memory construction. For each environment, we first collect trajectories by running the base agent on training tasks. Successful trajectories are converted into positive contracts. Each contract stores (i) a retrieval summary, (ii) a global applicability condition, and (iii) an ordered sequence

Environment	Task goal	Interaction interface	Metrics	Setting
ALFWorld	Household tasks such as object search, pick-and-place, heating, cooling, cleaning, examining, and placing two objects.	The agent receives a natural-language observation and emits one textual action from the ALFWorld action space.	Success rate.	Max steps: 20; episodes: 134; split: `test`.
ScienceWorld	Science procedures requiring ordered manipulation, state tracking, and intermediate verification.	The agent observes the current science-world state and emits one environment action at each step.	Official task score and success rate.	Max steps: 50; episodes: 100; split: `test`.
WebShop	Online shopping tasks in which the agent must search, inspect products, select required options, and purchase a matching item.	The agent observes page text plus available search/click actions and emits a single action such as `search[...]` or `click[...]`.	Mean reward and success rate.	Max steps: 15; episodes: 134; split: `test`.

Table 4: Environment details. The maximum number of steps follows the evaluation configuration used in our experiments.

Algorithm 2 Validity-Gated Routing

Require: Task t, current state s, positive memory bank $\mathcal{M}^+$, retriever $\mathcal{R}$, candidate size k, scoring agents $\{\mathcal{J}_m\}_{m=1}^{M}$
Ensure: Selected contract $e^\star$ and aligned chunks $\widetilde{C}$

$C_{\text{rel}} \leftarrow \textsc{TopK}(\mathcal{R}, t, \mathcal{M}^+, k)$
$C_{\text{app}} \leftarrow \emptyset$
for all $e_i^+ = \langle S_i, G_i, C_i \rangle \in C_{\text{rel}}$ **do**
  $v_i, r_i \leftarrow \textsc{CheckApplicability}(G_i, t, s)$
  if $v_i = \text{true}$ **then**
    $C_{\text{app}} \leftarrow C_{\text{app}} \cup \{e_i^+\}$
  end if
end for
if $C_{\text{app}} = \emptyset$ **then**
  return $\emptyset, \emptyset$ ▷ fall back to the base policy without contract reuse
end if
for all $e_i^+ \in C_{\text{app}}$ **do**
  for $m = 1, \ldots, M$ **do**
    $z_{im} \leftarrow \textsc{Score}(\mathcal{J}_m, e_i^+, t, s)$
  end for
  $u_i \leftarrow \textsc{Aggregate}(z_{i1}, \ldots, z_{iM})$
end for
$e^\star = \langle S^\star, G^\star, C^\star \rangle \leftarrow \arg\max_{e_i^+ \in C_{\text{app}}} u_i$
$\widetilde{C} \leftarrow \textsc{AlignChunks}(C^\star, t, s)$
return $e^\star, \widetilde{C}$

Algorithm 3 Contract-Aware Execution

Require: Task t, current state s, aligned chunks C_e, negative guard bank $\mathcal{M}^-$, executor policy π, monitor agent $\mathcal{V}$
Ensure: Execution history H

$H \leftarrow \emptyset$
$\Gamma \leftarrow \textsc{RetrieveGuards}(t, \mathcal{M}^-)$
for $\tilde{c}_j = \langle d_j, a_j, q_j \rangle \in C_e$ **do**
  $\kappa_j \leftarrow \emptyset$
  $\delta_j \leftarrow \emptyset$
  for each $\hat{a}_{j\ell} \in a_j$ **do**
    $x_{j\ell} \leftarrow (t, s, H, d_j, \hat{a}_{j\ell}, a_{j,\ell:L})$
    $x_{j\ell} \leftarrow \textsc{Augment}(x_{j\ell}, \Gamma, \kappa_j)$
    $u_{j\ell} \leftarrow \pi(x_{j\ell})$
    Execute $u_{j\ell}$; observe (o, s')
    $H \leftarrow H \circ [(s, u_{j\ell}, o, s')]$; $s \leftarrow s'$
    $m_{j\ell}, v_{j\ell} \leftarrow \mathcal{V}(d_j, q_j, H, o)$
    if $m_{j\ell} = \text{completed}$ **then**
      if $v_{j\ell} = \text{satisfied}$ **then**
        break
      else
        $\delta_j \leftarrow q_j$
        break
      end if
    end if
  end for
  if $\delta_j \neq \emptyset$ **then**
    $\kappa_j \leftarrow \text{LLMCorrect}(d_j, a_j, q_j, H, o)$
    $x_j \leftarrow \textsc{Augment}((t, s, H, d_j, q_j), \Gamma, \kappa_j)$
    $u_j \leftarrow \pi(x_j)$
    Execute u_j; observe (o, s')
    $H \leftarrow H \circ [(s, u_j, o, s')]$; $s \leftarrow s'$
  end if
end for
return H

of executable chunks. Each chunk contains a subgoal description, reference actions, and an expected postcondition. Failed trajectories are used only when the failed task can be localized to a clear cause. For each such task, we extract defensive rules from the failure feedback and store them together with the corresponding task description in the negative guard bank. During execution, EffectMem retrieves guards whose associated tasks are similar to the current task and injects their defensive rules into the executor context.

Retrieval and routing. Unless otherwise specified, EffectMem retrieves $k = 5$ candidate contracts for each test episode before applying validity-gated routing. In our implementation, the initial retriever uses cosine similarity over normalized task descriptions, and the applicability gate checks structured hard conditions extracted from the task metadata or instruction. For ALFWorld, these hard conditions include task type, required facilities, and target receptacles. For WebShop, they include price constraints and instruction attributes that should not be contradicted by the retrieved contract. The candidate contracts that pass applicability checking are then scored by multiple LLM-based assessors and aligned to the current task before execution.

Contract-aware execution. During online execution, EffectMem does not replay the full historical trajectory. Instead, it executes the aligned

Parameter	Value
Default retrieved contracts k	5
Random seed	42
ALFWorld maximum steps	20
ScienceWorld maximum steps	50
WebShop maximum steps	15
LLM temperature	0.3

Table 5: Core implementation parameters.

chunks one by one. Before execution, EffectMem retrieves task-relevant guards from the negative guard bank according to the current task description and injects their defensive rules into the executor context. After each chunk is executed, a monitor agent examines the executor's thinking history, executed actions, and environment feedback to determine whether the current subgoal has been completed. Once the subgoal is completed, the monitor verifies whether the expected postcondition is satisfied. If the expected effect is not achieved, the discrepancy is passed to an LLM corrector to generate temporary recovery guidance. This correction is used only within the current episode and is not written back to the memory banks during evaluation.

Model backends. The main comparison uses GPT-4-Turbo. Additional experiments use Qwen3-Max and Qwen3-8B to evaluate whether executable memory remains useful across model scales. For closed-source models, we call the corresponding chat completion API. For open-source backends, we use an OpenAI-compatible local serving endpoint. In Table 5, we have summarized all the hyperparameters according to the final experimental run. Table 6 summarizes the LLM backend used for each LLM-mediated operation in EffectMem.

Evaluation settings and comparison scope. Table 7 summarizes the evaluation settings of EffectMem and the compared methods, including the number of test tasks, maximum environment steps, and whether offline memory construction is used. The main comparison treats each method as a complete system under its reported configuration, rather than as a strictly equal-model or equal-token comparison. For EffectMem, memory construction is completed before evaluation, and the resulting memory banks remain fixed without accessing or updating any test trajectories. The model assignments for different EffectMem operations are reported in Table 6.

Method operation	Model used
Trajectory collection	Qwen3-Max
Successful-trajectory analysis	Qwen3-Max
Guard extraction	Qwen3-Max
Contract verification	Qwen3-Max
Multi-agent assessment	GPT-4-Turbo; Qwen3-Max; DeepSeek
Chunk alignment	Qwen3-Max
Online action generation	GPT-4-Turbo; Qwen3-Max; Qwen3-8B
Monitor Agent	Qwen3-Max
Discrepancy-guided correction	Qwen3-Max

Table 6: LLM-mediated operations in EffectMem and their model backends.

D Example of Noisy Memory

To illustrate how noisy memories are constructed, we provide a concrete noisy memory example in Table 8. Starting from the same clean temperature-measurement memory, we generate three noisy variants by perturbing different components of the trajectory. For global-condition noise, we corrupt the task-level tool-use condition by replacing the required thermometer with a stopwatch. Although the resulting memory is still related to scientific measurement, it is no longer applicable to the current temperature-measurement task because a stopwatch cannot satisfy the thermometer requirement. For chunk-level execution noise, we keep the measurement procedure unchanged but modify the final placement action from the correct yellow box to the incorrect orange box, causing the local execution effect to violate the task-specific decision rule. For failure-disguised noise, we remove the required thermometer-use action while preserving a success-like postcondition that claims the temperature has already been measured. These three transformations respectively test whether the agent can reject globally inapplicable memories, detect locally incorrect execution effects, and block failed trajectories disguised as successful memories.

E Prompt Templates

This section lists the prompt templates used by EffectMem for online action generation. Figures 5–8 should be read as environment-specific executor interfaces rather than as the whole memory mechanism: each template fixes the observation–

Method	Test tasks ALFWorld	Test tasks ScienceWorld	Test tasks WebShop	Max steps ALFWorld	Max steps ScienceWorld	Max steps WebShop	Offline memory construction
ReAct	134	100	134	20	80	15	No
ExpeL	134	100	134	20	80	15	Yes
AutoGuide	134	–	134	20	80	15	Yes
QuBE	134	–	134	20	80	–	No
ExpeL+QuBE	134	100	134	20	80	15	Yes
AutoRefine	–	30	–	–	50	–	Yes
KnowMap	–	Not specified	–	–	100	–	Yes
CFGM	134	100	134	20	80	15	Yes
EffectMem	134	100	134	20	50	15	Yes

Table 7: Comparison of evaluation settings across methods. A dash indicates that the corresponding environment was not evaluated, whereas "Not specified" indicates that the number of test tasks was not reported. Offline memory construction indicates whether a method constructs a memory bank before test-time evaluation.

Noise Type	Clean Memory	Noisy Memory	Targeted Capability
Global-condition noise	There is a thermometer used in the task.	There is a stopwatch used in the task.	Contract-level applicability checking / validity-gated routing.
Chunk-level execution noise	The agent measures the temperature as 13°C and moves unknown substance B to the yellow box, because 13°C is below 100.0°C.	The agent still measures the temperature as 13°C, but moves unknown substance B to the orange box.	Postcondition verification / chunk-level effect checking.
Failure-disguised noise	The agent first uses the thermometer to measure unknown substance B, and then places it into the yellow box.	The agent skips the thermometer measurement step but still claims that the temperature was measured as 13°C.	Negative experience guard / failure-aware prevention.

Table 8: Examples of noisy memory. The three noise types corrupt different aspects of memory reuse: global tool-use applicability, local execution effects, and pseudo-positive failure patterns.

action format, the legal action grammar, and the domain constraints that the agent must obey when it receives aligned chunks from validity-gated routing. During contract-aware execution, these base prompts are further augmented with the selected executable chunks, active negative guards, and temporary correction guidance described in Section 3.3.

ALFWorld prompt. Figure 5 shows the household-interaction prompt used for ALFWorld. The prompt emphasizes exact action formatting, household affordances, task-flow structure, and recovery from invalid feedback. These constraints match the role of EffectMem in ALFWorld: retrieved contracts provide reusable procedural structure, while the prompt keeps individual actions grounded in the current receptacles, objects, inventory state, and step budget.

ScienceWorld prompt. Figure 6 and Figure 7 present the ScienceWorld prompt templates used for ScienceWorld, where success depends on performing ordered scientific procedures and verifying intermediate state changes. The prompt therefore serves as a domain interface for effect-verifiable reuse: it asks the agent to act according to the current observation and available operations, while EffectMem supplies the selected contract and checks whether each aligned chunk achieves its expected postcondition.

WebShop prompt. Figure 8 presents the WebShop prompt, which differs from the two embodied-style environments because actions are page operations such as searching, clicking products, selecting options, and purchasing. The prompt therefore makes attribute extraction, option tracking, action validity, and step efficiency explicit. This is important for EffectMem because many shopping trajectories are semantically similar but differ in price limits, required attributes, or already-selected options.

F Examples

F.1 Correction Example

Figure 9 illustrates how EffectMem handles a failed effect check during contract-aware execution. The example complements the static prompt templates above: prompts define the legal action interface, whereas discrepancy-guided correction is invoked only after an aligned chunk fails to produce its expected postcondition. The correction is local to the current episode and does not update either the

Interact with a household to solve ALFWorld tasks with minimal steps. Inputs include initial environment description, task goal, and real-time latest observation per turn.
CRITICAL OUTPUT FORMAT: Output exactly one line starting with Action: no reasoning, tags, extra text or blank lines.
GENERAL PRINCIPLES: Prioritize task completion over exploration; judge by latest observation, inventory and past feedback; treat ineffective actions as failure and adjust strategies; use exact standard object names; avoid redundant navigation and useless operations.
ACTION SPACE: go to {recep}, take {obj} from {recep}, move {obj} to {recep}, open/close {recep}, clean/heat/cool/slice {obj} with {recep}, examine {obj/recep}, use {obj}, inventory.
FAILURE RECOVERY: Do not repeat failed actions; adjust location, receptacle or action type timely; verify object holding state, position and receptacle status when failing; backtrack to missing prerequisites for persistent errors.
OBJECT & EFFICIENCY RULES: Use object/receptacle names exactly as observed; prohibit redundant operations; use inventory only for ambiguous holding state. Final output: strictly output one line starting with Action:

Figure 5: ALFWorld action-generation prompt. The template constrains the agent to output exactly one `Action:` line from the household action space and encodes domain affordances for navigation, object pickup, placement, opening, cleaning, heating, cooling, slicing, examination, toggling, and inventory checks. It also specifies precondition rules and recovery behavior after invalid feedback such as `Nothing happens.` These prompt constraints complement EffectMem's executable contracts by keeping each reused chunk grounded in valid household actions and current-state preconditions.

positive contract bank or the negative guard bank during evaluation.

F.2 Guard Example

Figure 10 illustrates how EffectMem extracts a negative guard from a failed trajectory. In this example, the agent successfully obtains the thermometer and reaches the target object, but skips the required pick-up step before attempting to use the thermometer. The repeated invalid use action causes the episode to reach the maximum step limit with zero score. EffectMem analyzes the failed trajectory, identifies the missing pick-up step as the root cause, and distills it into a reusable guard rule. The extracted guard is then stored in the negative guard bank and can be retrieved in later similar tasks to prevent the same missing-prerequisite failure.

F.3 Monitor Agent Example

Figure 11 illustrates monitor-based postcondition verification during contract-aware execution. After the executor produces a thought indicating that the current subgoal has been completed, the monitor agent examines the executor's thinking history, executed actions, and latest environment feedback. It then compares the observed outcome with the expected postcondition of the current chunk. In this example, the expected postcondition requires the wood to be in the inventory, but the execution history indicates that the wood has not been picked up. The monitor therefore marks the postcondition as unsatisfied and triggers discrepancy-guided correction.

You are an agent acting in a text-based science environment. Your job is to output exactly one valid next action and nothing else.

General rules: 1. Output only one action with no explanation. 2. Only use explicitly mentioned objects, containers, substances, rooms and doors. 3. Do not assume inventory objects without explicit observation confirmation. 4. Do not assume room accessibility without visible connecting doors. 5. Avoid repeated invalid action syntax after syntax mismatch failure. 6. Stop retrying failed action patterns twice without environment state changes. 7. Prefer short, simple, environment-grounded commands. 8. Avoid unnecessary manipulations that may undo achieved progress.

Internal Tracking Requirements: Before each action, track current room, visible doors, inventory contents, target object/room/tool, completed subgoals, and target object location state.

Navigation rules: 1. Execute "go to <room>" only for currently visible rooms. 2. Move step-by-step via intermediate rooms for unreachable target rooms. 3. Avoid repeated attempts to access unreachable rooms.

Interaction rules: 1. Only focus on objects in current room or inventory. 2. Only use tools with explicit accessibility (inventory or current scene).

Recovery rules: 1. Recover from failure via look around, check inventory, move to reachable room or pick up missing tools; prefer valid successful syntax for subgoal advancement. 2. Avoid infinite loops of repeated invalid exploration or movement actions.

Core Constraints: Fulfill all tool and setup preconditions before target operation. Stop blind execution and reset sequence if score drops sharply. Only place objects in currently visible containers.

Task interpretation rules: Determine subgoal order via task description, not assumed accessibility. Resolve unavailable prerequisites first for unfinished steps. Follow official valid action syntax strictly. Only focus on or use accessible objects. Prioritize action validity over task speediness.

Figure 6: ScienceWorld action-generation prompt. The template frames each episode as a stateful scientific procedure in which the agent must use the current observation to choose a valid next action, preserve the order of prerequisite operations, and avoid actions whose effects are not supported by the current state. This design is aligned with the ScienceWorld ablation setting in the main text, where ordered subgoals, state-dependent applicability, and postcondition verification are central to measuring validity-gated routing and contract-aware execution.

{System Prompt}
CRITICAL GUARDS FOR TASK SUCCESS:
{Guards}

Reference examples are lower priority than the current valid_actions, current observation, and current task order. Use them only for very general strategy, never for exact action copying. If any example conflicts with the current task order or current valid_actions, ignore the example.
{few_shot}

Current observation:
{observation}

Task description:
{task_description}

Valid actions:
{valid_actions_str}

Choose the single best next action.
Remember:
- Identify the earliest unfinished task-required subgoal from Task description.
- Treat Valid actions as the only allowed source of action syntax.
- Avoid repeating actions that did not change the state.
- Prefer the most task-relevant action, including prerequisite-checking, tool-acquisition, and route-building actions when needed.
Output exactly in this format:
Thought:
Action:

Figure 7: Guard-augmented ScienceWorld action-generation prompt. The template injects retrieved guards and reference examples into the current observation, task description, and valid-action list. It instructs the executor to identify the earliest unfinished subgoal, choose only from the valid actions, and treat retrieved examples as lower-priority guidance when they conflict with the current task order or valid action set. This prompt supports contract-aware execution by combining reusable memory guidance with current-state action constraints.

You are a shopping agent on an online store (WebShop). Your goal is to find and purchase the product that best matches the given instruction. Each turn receives current page observation and available actions, then outputs exactly one valid action.
ACTION SPACE: Two valid action types. **search[keywords]**: Search products with instruction-extracted keywords, only available on search pages with a search bar. **click[value]**: Click buttons/links using exact values from provided action list; valid values include product ASINs, detail tabs, attribute options, "buy now", "back to search", "next >", "< prev".
OUTPUT FORMAT: Output exactly one line per turn in fixed format: Action: search[your keywords] / Action: click[exact value]. No extra text or explanations.
CORE TASK: Extract key information from user instruction, including item type, required attributes (color/size/material), price upper limit, and special functional requirements.
STATE TRACKING (CRITICAL): Track all selected attributes (color/size/options) in real time. Unchanged page options after clicking mean successful selection. Never click the same option twice consecutively or repeatedly select confirmed options.
DECISION WORKFLOW: **1. Search**: Use concise keyword combinations; avoid duplicate searches; optimize results via page turning or adjusted keywords. **2. Select Product**: Click promising ASINs, filter out over-priced products, and never revisit rejected products. **3. Select Options**: Select all required attributes once only, confirm successful selection via page state.**4. Verify & Purchase**: Check full attribute matching and price compliance before clicking "buy now".
CRITICAL RULES: No repeated option clicks; no duplicate consecutive searches; no revisiting rejected products; no purchase without full option selection; always track selected attributes to avoid redundant operations.
IMPORTANT SPEC: Only use valid click values from the official available actions list, do not invent values. Record viewed products, selected options and remaining steps mentally. Output only standard Action line with no reasoning content.
Final Output: Strictly output one line starting with Action:

Figure 8: WebShop action-generation prompt. The template restricts each response to one valid `search[...]` or `click[...]` action, requires click values to be copied from the available actions, and asks the agent to extract item type, required attributes, price constraints, and special requirements from the instruction. It further tracks selected options, discourages repeated searches or repeated option clicks, and permits `buy now` only after the product attributes and required options have been verified. These rules make the prompt compatible with EffectMem's applicability checks, which reject retrieved shopping trajectories that look relevant but contradict the current purchase constraints.

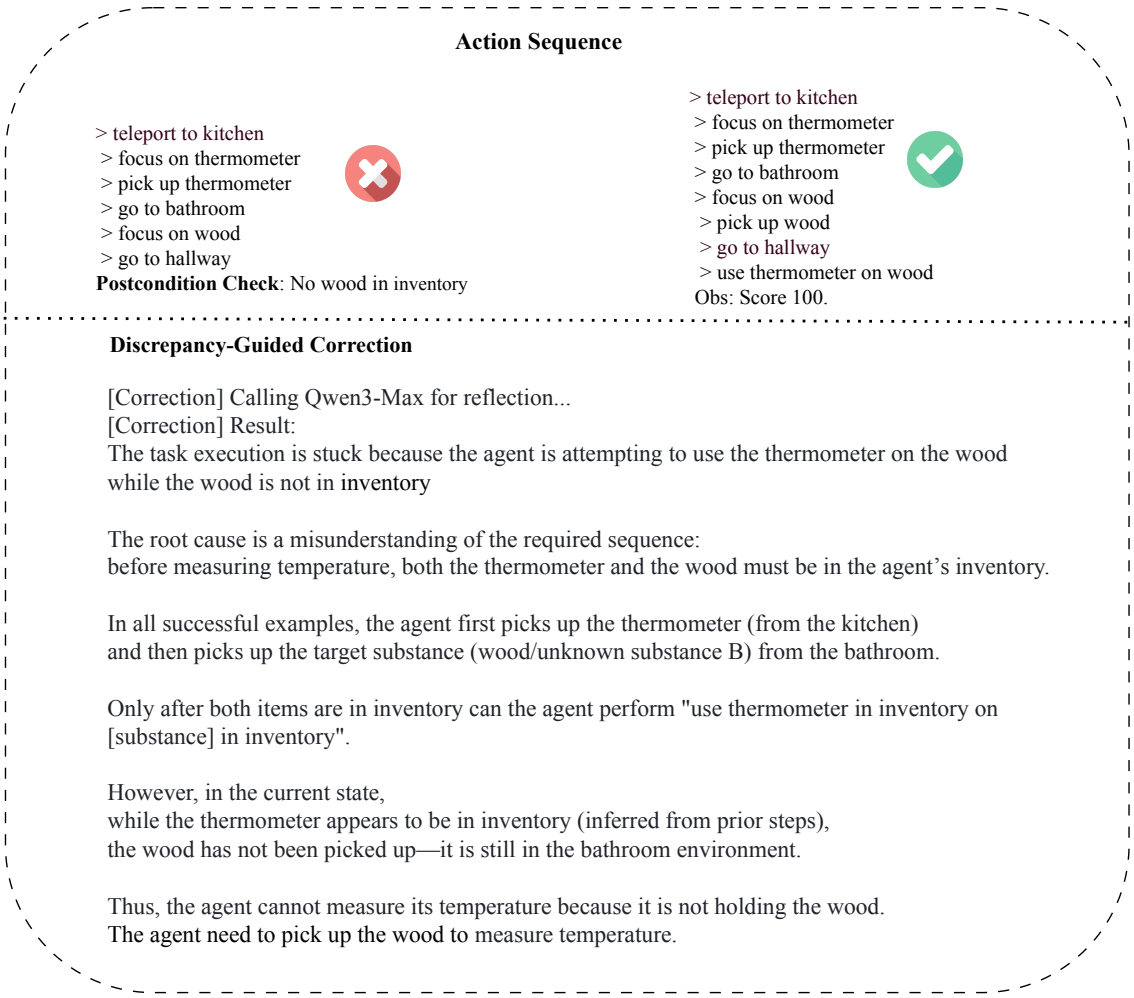


Figure 9: Representative discrepancy-guided correction example. After an aligned chunk is executed, the monitor agent examines the execution history and latest observation to verify whether the chunk's expected postcondition has been satisfied. When the expected effect is not achieved, the corrector receives the current task, execution history, selected chunk, unmet condition, and environment feedback, then produces temporary recovery guidance for the next action. This example illustrates how contract-aware execution turns postcondition failure into localized repair instead of continuing to replay a trajectory whose assumptions no longer hold.

Action Sequence

> teleport to kitchen
> focus on thermometer
> pick up thermometer
> go to bathroom
> focus on wood
> go to hallway
> use thermometer on wood
> use thermometer on wood
(Reached Max Steps)
Obs: Score 0.

[Guard] Calling Qwen3-Max for Extracting...
[Guard] Result:

1. Analyze the failed action sequence and Score 0 result.

2. Identify the root cause: The agent skipped the pick up wood step after focusing on the target, then repeatedly executed the invalid use thermometer on wood action, leading to wasted steps and task failure.

3. Generalize the pattern into a reusable guard rule: All interactive use operations require prior pick up of the target object; skipping this step will always result in invalid behavior.

4. Formulate the final guard: *Do not skip pick up step.*

Guards: Do not skip pick up step.

Figure 10: Guard extraction example. EffectMem analyzes a failed temperature-measurement trajectory in which the agent skips the required pick-up step before using the thermometer. The LLM-based analyzer identifies this missing prerequisite as the root cause and converts it into a reusable negative guard, *Do not skip pick up step*. This guard can later constrain execution in similar tasks and prevent repeated invalid use actions.

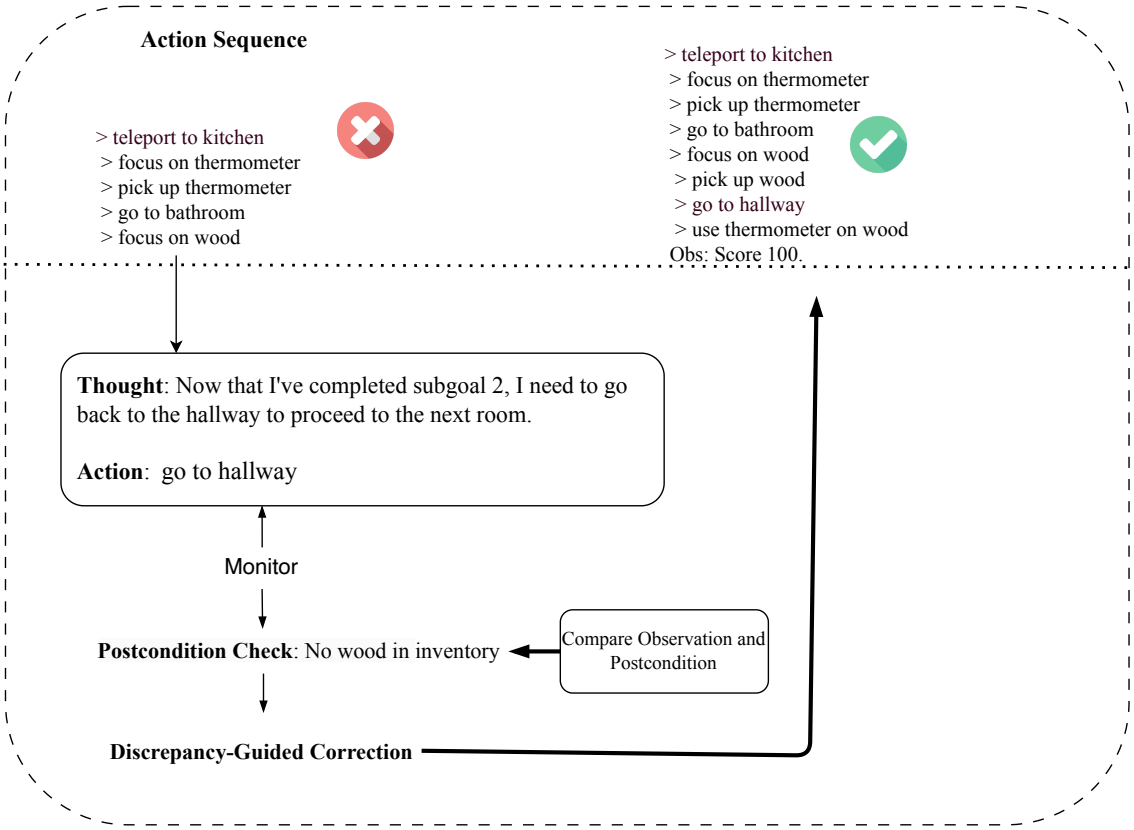


Figure 11: Monitor-agent verification example. After the executor assumes that the current subgoal has been completed, the monitor agent checks the thinking history, executed actions, and latest observation against the expected postcondition. Here, the postcondition requires the wood to be in the inventory, but the execution history does not include the required pick-up action. The monitor therefore detects an unmet postcondition and triggers discrepancy-guided correction.