EVOCOG: COMPOSABLE COGNITIVE PROGRAMS FOR ADAPTIVE REASONING IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language-model-based agents have demonstrated remarkable capabilities in complex decision-making and tool-use tasks. However, existing adaptive reasoning methods typically adapt computation within a predefined set of reasoning modes, limiting their ability to optimize and evolve the structured cognitive procedures required across diverse interactive environments. In this paper, we introduce EvoCog, a framework that represents an agent's cognitive space as executable programs built from four typed operations: GROUND, PLAN, CHECK, and ACT. A causal grammar constrains dependencies among these operations, defining a space of structurally valid programs. EvoCog learns and evolves these programs through a three-stage pipeline. First, program-conditioned supervised fine-tuning teaches a shared executor to follow assigned programs while preserving expert actions. Second, Sparse Counterfactual Program Optimization (SCPO) optimizes task execution and a cost-aware router for step-level program selection. Finally, EvoCog supports test-time adaptation by iteratively repairing its cognitive programs from interaction failures, without updating model parameters. Experiments on ALFWorld, ScienceWorld and WebShop demonstrate that EvoCog improves interactive task solving while reducing inference cost. With Qwen2.5-7B, it reaches a 77.47% success rate, outperforming GPT-5(+26.57%) and GRPO(+15.1%), while using 51.4% fewer tokens.

1 INTRODUCTION

LLM-based agents are increasingly deployed and interact with diverse environments, such as web pages, embodied worlds, and external tool interfaces (Yao et al., 2023; Shridhar et al., 2021; Wang et al., 2022; Yao et al., 2022; Schick et al., 2023). Unlike single-turn language tasks, interactive environments unfold over multiple steps, and different states can impose different cognitive demands (Yang et al., 2026a). An agent may need to ground the current observation, plan toward a subgoal, verify a previous transition, or act directly. Effective interaction therefore requires deciding not only what action to take, but also how to reason before taking it.

Recent work has begun to make this second decision adaptive. Some methods adjust the amount of reasoning allocated to each step, selecting different reasoning efforts according to the current state (Yang et al., 2026a; Graves, 2016; Wei et al., 2022). Others select among predefined reasoning modes or cognitive strategies during problem solving (Jiang et al., 2026). CogRouter (Yang et al., 2026b), for example, learns to route each step among several cognitive levels ranging from direct action to more deliberate reasoning. However, these alternatives remain predefined as complete modes whose internal reasoning functions are coupled. This becomes restrictive when states require qualitatively different *combinations* of cognition rather than simply more or less reasoning. Figure 1 shows that preferred procedures vary across matched states and that programs with similar computational cost can yield substantially different utilities. We therefore ask: *can cognition itself be represented as an executable and evolvable program rather than a fixed reasoning mode?*

To address this problem, we introduce **EvoCog**, which represents cognition as executable programs composed from four reusable operations: GROUND, PLAN, CHECK() and ACT. A causal grammar specifies valid dependencies among these operations, yielding programs such as GROUND→ACT, PLAN→CHECK→ACT, and CHECK→PLAN→ACT. This factorization changes the unit of adapta-

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.

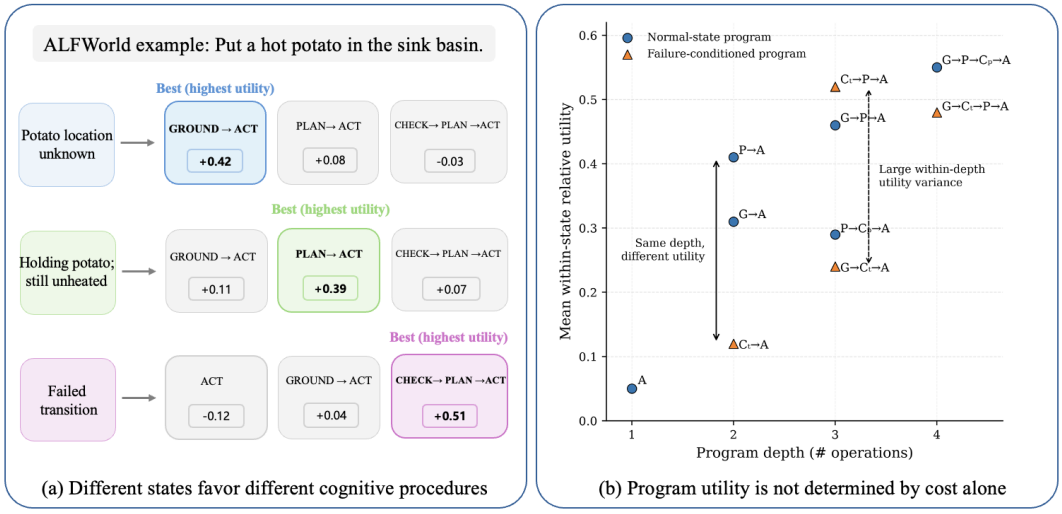

Figure 1: **Cognitive demand depends on program structure, not only reasoning depth.** **(a)** Matched-state counterfactual evaluations show that different states favor different cognitive procedures. **(b)** Programs with the same operation count can yield substantially different utilities. G, P, C_p, C_t, and A denote GROUND, PLAN, CHECK(PLAN), CHECK(LAST TRANSITION), and ACT.

tion from a complete reasoning mode to a structured procedure whose components can be reused and composed.

A compositional cognitive space requires the agent to learn two distinct abilities: executing a specified program and selecting an appropriate one. EvoCog separates these objectives. Program-conditioned supervised fine-tuning first teaches a shared executor to follow externally assigned programs while preserving expert actions, without assigning an optimal program label to each state. We then introduce **Sparse Counterfactual Program Optimization (SCPO)** to learn state-dependent cognitive control. SCPO branches alternative programs from a small number of matched interaction states and compares their downstream utilities, providing direct credit for program selection while jointly optimizing task execution and reasoning cost. Finally, EvoCog allows cognition to remain editable after training. With all model parameters frozen, a failed rollout triggers failure attribution and a small set of grammar-constrained residual modifications attached to existing programs. The residual-augmented variant is retested on the same task, and failed retests provide evidence for subsequent repair rounds. Residual modifications are added to a persistent cognitive library for later reuse. Test-time interaction therefore updates the agent's explicit cognitive library rather than its model parameters.

We evaluate EvoCog on ALFWorld (Shridhar et al., 2021), ScienceWorld (Wang et al., 2022) and WebShop (Yao et al., 2022). With Qwen2.5-7B, EvoCog achieves an average success rate of 77.47% before test-time evolution, outperforming GPT-5(+26.57%) and GRPO(+15.1%), while using 51.4% fewer reasoning tokens. Test-time cognitive evolution further improves the frozen agent

Our contributions are three-fold:

- We introduce a typed cognitive program representation that composes reusable GROUND, PLAN, CHECK, and ACT operations under a causal grammar, moving adaptive reasoning beyond selection among predefined complete modes.
- We propose program-conditioned supervised fine-tuning and **Sparse Counterfactual Program Optimization (SCPO)** to separate program execution from state-dependent program selection and assign counterfactual credit to cognitive choices.
- We introduce failure-guided test-time cognitive evolution, where a frozen agent refines cognitive programs from interaction failures and promotes only independently verified repairs into a persistent program library without updating model parameters.

2 Related Work

Adaptive Reasoning for LLM Agents. Adaptive reasoning aims to allocate inference computation according to changing task demands (Graves, 2016). Ares dynamically adjusts reasoning effort during interaction (Yang et al., 2026a), while Chain of Mindset switches among predefined reasoning strategies (Jiang et al., 2026). Most closely related to our work, CogRouter learns step-level routing among hierarchical cognitive levels ranging from direct action to more deliberate reasoning (Yang et al., 2026b). These approaches demonstrate the value of state-dependent cognition, but the adaptation units remain predefined cognitive configurations. EvoCog instead factorizes cognition into typed reusable operations and treats their composition as the unit of adaptation.

Reinforcement Learning for LLM Agents. Reinforcement learning has become an important approach for improving LLM agents through interaction with environments (Ouyang et al., 2022; Rafailov et al., 2023). Existing methods commonly optimize complete trajectories using task rewards, including PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024). For long-horizon tasks, however, trajectory-level rewards provide limited information about which intermediate decisions are responsible for success or failure. Recent work therefore introduces finer-grained credit assignment. GiGPO (Feng et al., 2025) augments episode-level optimization with step-level comparisons at repeated interaction states, while CogRouter (Yang et al., 2026b) applies reinforcement learning to step-wise cognitive-level selection. EvoCog addresses a different decision space by assigning counterfactual credit directly to cognitive program selection.

Test-Time Agent Adaptation and Self-Improvement. Several methods improve agents from interaction experience without updating model parameters. Reflexion stores verbal feedback from previous failures, while ExpeL extracts reusable insights from accumulated experience (Shinn et al., 2023; Zhao et al., 2024). Other approaches induce structured external knowledge, such as guidelines or reusable workflows (Fu et al., 2024; Wang et al., 2025). RESKILL maintains explicit failure hypotheses and candidate skill repairs through environment retesting (Deng et al., 2026). EvoCog differs in the adapted object: it evolves typed cognitive programs constrained by the same grammar used during training, leaving the learned executor and model parameters unchanged.

3 EvoCog

3.1 Overview and Problem Formulation

We consider an LLM agent interacting with an environment over a sequence of decision steps. At step t, the agent receives a decision context

$$x_t = (g, o_t, h_t, \mathcal{A}_t, f_t),$$

where g is the task instruction, o_t is the current observation, h_t denotes the interaction history, $\mathcal{A}_t$ is the set of currently available actions and f_t contains explicit failure feedback when present. The agent only accesses information available at decision time.

EvoCog augments the agent with an explicit cognitive program library $\mathcal{L}$. Given a decision context x_t, the grammar and state constraints define a valid subset $\mathcal{S}(x_t) \subseteq \mathcal{L}$. The Router selects a program only from this valid support:

$$P_t \sim \pi_\theta(P|x_t), \qquad P \in \mathcal{S}(x_t),$$

and a shared Executor follows the selected program to generate cognitive states and an environment action:

$$(y_t, a_t) \sim \pi_\theta(\cdot|x_t, P_t).$$

This separation allows EvoCog to optimize not only how an action is generated, but also which cognitive procedure should precede it.

EvoCog learns and evolves this cognitive system in three stages. An overview of the framework is shown in Figure 2. Stage I learns the execution contract of assigned cognitive programs through program-conditioned SFT. Stage II optimizes program selection and execution within a fixed program space using reinforcement learning. Stage III freezes the model parameters and extends adaptation beyond selecting fixed programs by updating the cognitive library with verified residual modifications at test time.

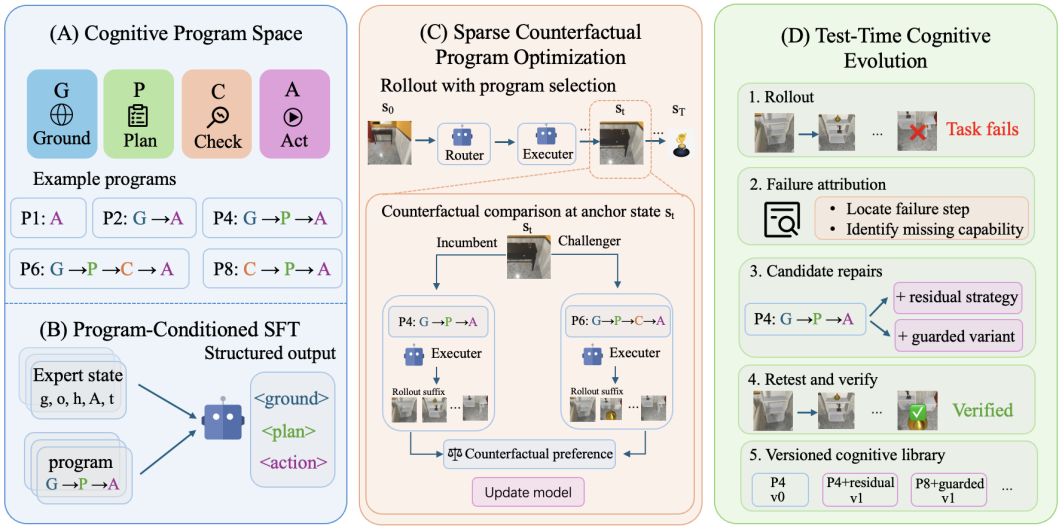

Figure 2: **Overview of EvoCog.** EvoCog represents cognition as composable programs over four typed operations. (A) defines the cognitive program space. (B) learns program execution through program-conditioned SFT. (C) optimizes program selection via Sparse Counterfactual Program Optimization. (D) evolves the cognitive library at test time through failure-guided repair and verification.

3.2 Cognitive Program Space

EvoCog represents cognition using four reusable operations: Ground ($\mathcal{G}$), Plan ($\mathcal{P}$), Check ($\mathcal{C}$), and Act ($\mathcal{A}$). Each operation has a narrow execution contract: $\mathcal{G}$ extracts task-relevant information, $\mathcal{P}$ specifies intermediate goals, $\mathcal{C}$ verifies plans or failed transitions, and $\mathcal{A}$ emits exactly one environment action. Each operation follows a narrow execution contract to prevent one component from implicitly replacing another.

Programs are generated under a causal grammar rather than arbitrary combinations. Specifically, act always terminates a program, plan checking follows a preceding plan, and transition checking requires explicit failure evidence. Under these constraints, EvoCog constructs a compact seed library of ten programs covering normal interaction and recovery scenarios (see Appendix A.2.2).

The resulting P1–P10 library defines the initial cognitive program space used by both SFT and RL. Unlike fixed reasoning-depth levels, these programs differ in the type and ordering of cognitive operations they invoke, enabling state-dependent composition of cognition.

3.3 Stage I: Program-Conditioned Supervised Fine-Tuning

Before learning which cognitive program to select, the model must first learn to execute different programs reliably. We therefore initialize EvoCog with program-conditioned supervised fine-tuning. Starting from expert trajectories, each interaction step provides a state x_t and an expert action a_t^*. A grammar-valid program P_t is externally assigned from the legal set $\mathcal{S}(x_t)$:

$$P_t \sim q(P \mid x_t), \qquad P \in \mathcal{S}(x_t),$$

where q is a balanced sampler that provides coverage over the initial program library. Importantly, P_t is an input condition rather than a prediction target: Stage I trains the model to execute a given cognitive program, not to decide which program should be used.

For each assigned program, a teacher model constructs the corresponding structured cognitive trace while preserving the expert action. We denote the resulting target as

$$y_t^P = [P_t,\ c_t^P,\ a_t^*],$$

where c_t^P contains only the cognitive operations required by P_t and follows their prescribed order. This produces the program-conditioned dataset

$$\mathcal{D}_{\mathrm{SFT}} = \left\{ (x_t, P_t, y_t^P) \right\}.$$

A shared executor is optimized with the standard autoregressive objective:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{(x,P,y)\sim\mathcal{D}_{\mathrm{SFT}}} \left[\sum_{k=1}^{|y|} \log \pi_\theta \left(y_k \mid x, P, y_{<k} \right) \right].$$

This stage establishes a stable mapping from a specified program to its corresponding reasoning-and-action behavior. It does not optimize $p(P \mid x)$; state-dependent program selection is learned only in Stage II.

3.4 Stage II: Sparse Counterfactual Program Optimization

After Stage I, the executor can follow different cognitive programs, but it does not yet know which program is appropriate for a given interaction state. We therefore introduce **Sparse Counterfactual Program Optimization (SCPO)** to jointly optimize program execution and cognitive-program selection.

At each step t, the Router selects a program from the state-dependent legal subset $\mathcal{S}(x_t)$,

$$P_t \sim \pi_\theta(P \mid x_t), \qquad P \in \mathcal{S}(x_t),$$

and the Executor subsequently generates the corresponding cognitive trace and environment action,

$$(y_t, a_t) \sim \pi_\theta(\cdot \mid x_t, P_t).$$

Trajectory utility. For each training task, we sample a group of complete interaction trajectories $\{\tau_i\}_{i=1}^K$. Rather than optimizing task reward alone, we define a cost-aware trajectory utility

$$U(\tau) = R_{\mathrm{task}}(\tau) - \lambda C(\tau),$$

where $C(\tau)$ includes reasoning-token usage, interaction steps, and invalid execution costs. This objective encourages the agent to balance task performance and cognitive cost rather than simply favoring longer reasoning procedures.

Sparse counterfactual program credit. Trajectory-level rewards cannot distinguish whether a failure is caused by an inappropriate program choice or by incorrect execution of the selected program. SCPO therefore introduces sparse counterfactual comparisons at informative anchor states. Anchor states are selected using predefined failure-sensitive criteria, including invalid execution, explicit environment failure feedback, and stalled progress. Instead of evaluating all programs at every step, SCPO compares the incumbent program selected in the original rollout with one challenger program selected from the remaining grammar-valid alternatives.

Given the program selected in the original rollout, P_{inc}, SCPO evaluates one alternative legal program P_{chal} from the same state:

$$\Delta(s) = \hat{U}(s, P_{\mathrm{inc}}) - \hat{U}(s, P_{\mathrm{chal}}),$$

where $\hat{U}(s, P)$ denotes the estimated downstream utility obtained by executing program P from state s. The incumbent utility is obtained from the original trajectory, while the challenger is evaluated through one additional suffix rollout after restoring the same environment state. This sparse branching strategy provides direct credit for cognitive program selection without requiring exhaustive program evaluation.

Joint policy optimization. SCPO uses ordinary rollouts to optimize the Executor and Router, while counterfactual comparisons provide an additional preference signal for the Router. The overall objective is:

$$\mathcal{L}_{\mathrm{SCPO}} = \mathcal{L}_{\mathrm{exec}} + \alpha \mathcal{L}_{\mathrm{router}} + \lambda_{\mathrm{cf}} \mathcal{L}_{\mathrm{cf}},$$

where $\mathcal{L}_{\mathrm{cf}}$ is a utility-weighted pairwise preference objective over counterfactual program comparisons.

The Executor therefore learns how to realize a selected cognitive program, whereas the Router learns which program is preferable under different interaction states. Additional implementation details are provided in Appendix A.2.

3.5 Stage III: Test-Time Cognitive Evolution

Stage II optimizes program execution and selection within a fixed cognitive library space, but the realization of existing programs remains unchanged after training. Test-time failures may expose limitations that cannot be corrected by selecting another existing program, motivating adaptation of program execution behavior. EvoCog therefore introduces test-time cognitive evolution, where the model parameters remain frozen and the cognitive library $\mathcal{L}$ is updated through verified residual modifications.

Failure-guided repair. Given a failed trajectory, an external diagnosis module analyzes the failure and determines whether it results from program selection, execution, or a cognitive gap in the existing procedure. Only failures attributed to cognitive gaps trigger evolution. For an affected root program P_j, the repair module generates a guarded residual:

$$r_j = (g_j, \delta_j),$$

where g_j specifies when the residual is applicable and δ_j defines a local strategy modification. The residual augments the execution of the existing program without introducing new programs or changing the Router's action space:

$$P'_j = P_j \oplus (g_j, \delta_j),$$

Verified evolution. A candidate residual is temporarily installed and evaluated by replaying the same task with the frozen agent. It is added to the global cognitive library only if it improves task outcomes. Verified residuals persist and can be reused in later tasks when their guard conditions are satisfied. Thus, test-time evolution refines cognitive programs through reusable modifications without updating model parameters.

4 Experiments

4.1 Experimental Setup

Evaluation Protocol. We evaluate EvoCog on three interactive environments: ALFWorld (Shridhar et al., 2021), ScienceWorld (Wang et al., 2022), and WebShop (Yao et al., 2022). Further details can be found in Appendix A.1. Models are trained independently on each environment while using the same cognitive program space and training pipeline. We report success rate for all environments, together with environment score for ScienceWorld and WebShop and generated tokens as an inference-cost metric. All trainable methods use the same backbone, task split, interaction horizon, and rollout budget whenever applicable. For test-time cognitive evolution, model parameters remain frozen and only the external cognitive library is updated through verified residual modifications. Detailed training hyperparameters are provided in Appendix A.3.

Baselines. We compare EvoCog against three categories of baselines. *Prompting-based methods* include ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2023), Plan-and-Act (Erdogan et al., 2025) and ReflAct (Kim et al., 2025). *Adaptive-reasoning methods* include AdaptThink (Zhang et al., 2025) and CogRouter (Yang et al., 2026b). *Reinforcement-learning methods* include GRPO (Shao et al., 2024) and GiGPO (Feng et al., 2025), which provide trajectory-level and step-level policy optimization approaches(see Appendix A.6 for implementation details).

To isolate the contribution of SCPO, GRPO and GiGPO are additionally initialized from the same program-conditioned SFT checkpoint. For test-time evolution analysis, we apply the same evolution procedure to SCPO and CogRouter checkpoints to examine whether cognitive evolution transfers across different representations.

4.2 Main Results

Table 1 reports the first-pass performance of all methods on the Qwen2.5-7B-Instruct (Qwen et al., 2025) before test-time cognitive evolution. EvoCog achieves strong performance across ALFWorld, ScienceWorld, and WebShop, demonstrating the effectiveness of SCPO for adaptive reasoning across interactive environments. Results with Llama3.1-8B-Instruct (Dubey et al., 2024) are provided in Appendix A.4.

Table 1: Main results on ALFWorld, ScienceWorld, and WebShop. SR denotes success rate (%) and Tok. denotes generated tokens. Best results within each backbone are highlighted.

Method	ALFWorld		ScienceWorld			WebShop			Average	
	SR↑	Tok.↓	Score↑	SR↑	Tok.↓	Score↑	SR↑	Tok.↓	SR↑	Tok.↓
Frontier Large Language Models										
GPT-5	78.5	4260.5	72.2	35.5	15182.0	49.6	38.7	4762.3	50.90	8068.3
Gemini 2.5 Pro	70.0	4896.5	65.2	44.0	8942.4	**62.0**	31.66	5127.2	48.55	6322.0
Claude Sonnet 4.6	**88.5**	1275.3	**88.6**	**79.5**	1003.0	58.3	**41.9**	2152.2	**69.97**	1476.8
DeepSeek V3.2	66.5	171.2	62.4	54.5	1013.4	55.0	31.55	1627.4	50.85	937.3
Qwen2.5-7B-Instruct										
Direct	14.0	287.0	11.4	4.0	719.0	26.4	7.8	381.0	8.60	462.3
ReAct	31.5	627.3	8.0	2.5	1645.1	46.2	19.5	1584.3	17.83	1285.6
Reflexion	42.5	2340.8	10.1	2.5	3058.5	58.1	28.8	1365.5	24.60	2254.9
Plan-and-Act	23.5	245.97	9.7	2.0	564.3	12.6	3.4	300.8	9.63	370.4
ReflAct	37.5	869.5	6.0	8.5	2490.6	28.6	10.2	598.2	18.73	1319.4
AdaptThink	81.5	1372.1	62.5	44.5	1462.4	64.7	42.5	1524.9	56.17	1453.1
CogRouter	**92.5**	1739.4	84.6	72.0	1543.3	75.4	**62.0**	1723.6	75.50	1668.8
Program-SFT	56.5	802.2	59.6	36.5	2946.5	52.4	35.8	1696.1	42.93	1814.9
+ GRPO	82.5	885.1	67.3	49.0	1982.6	70.2	55.6	1927.3	62.37	1588.3
+ GiGPO	91.5	625.8	70.6	56.5	2062.5	75.4	62.0	1064.5	70.00	1250.9
+ SCPO (Ours)	**92.5**	537.8	**86.8**	**76.5**	957.1	**79.4**	61.4	820.3	**77.47**	771.7

Table 2: Test-time cognitive evolution on Qwen2.5-7B-Instruct.

Method	ALFWorld		ScienceWorld		WebShop	
	Gain	SR	Gain	SR	Gain	SR
SCPO+evo	+2.0	94.5	+2.0	78.5	+3.1	64.5
CogRouter+evo	+0.0	92.5	+4.5	76.5	+3.3	65.3

EvoCog enables adaptive cognitive control. EvoCog consistently improves interactive decision-making across all three benchmarks, significantly outperforming both open-source baselines and proprietary frontier LLMs. On Qwen2.5-7B, EvoCog achieves a 92.5% success rate on ALFWorld, a ScienceWorld score of 86.8 with 76.5% success rate, and a WebShop score of 79.4 with 61.4% success rate. These results show that EvoCog can learn state-dependent cognitive control by jointly optimizing program execution and program selection.

SCPO provides effective cognitive program optimization. Starting from the same program-conditioned SFT initialization, SCPO consistently outperforms GRPO and GiGPO. While standard RL methods optimize trajectory outcomes, SCPO introduces sparse counterfactual comparisons that assign credit directly to cognitive program choices. SCPO improves the average success rate from 42.93% with Program-SFT to 77.47%, while reducing average generated tokens from 1814.9 to 771.7 (57.5% reduction). Compared with GRPO and GiGPO, SCPO achieves higher success rates with 51.4% and 38.3% fewer tokens, respectively. These results demonstrate that SCPO improves both program selection quality and the performance-cost trade-off.

Test-time cognitive evolution improves frozen agents. Although SCPO learns an effective policy within the initial cognitive library, the fixed program space may still encounter unseen failure modes after deployment. Table 2 evaluates Stage III from the corresponding trained checkpoints. Test-time cognitive evolution keeps model parameters frozen and updates only the cognitive library through verified residual modifications. In our implementation, GPT-4o (OpenAI, 2024) serves as an external failure diagnosis and residual generation module, while the agent remains responsible for execution. Starting from the SCPO checkpoint, EvoCog obtains additional gains across all three environments, improving first-pass performance through persistent cognitive updates. Moreover, applying the same

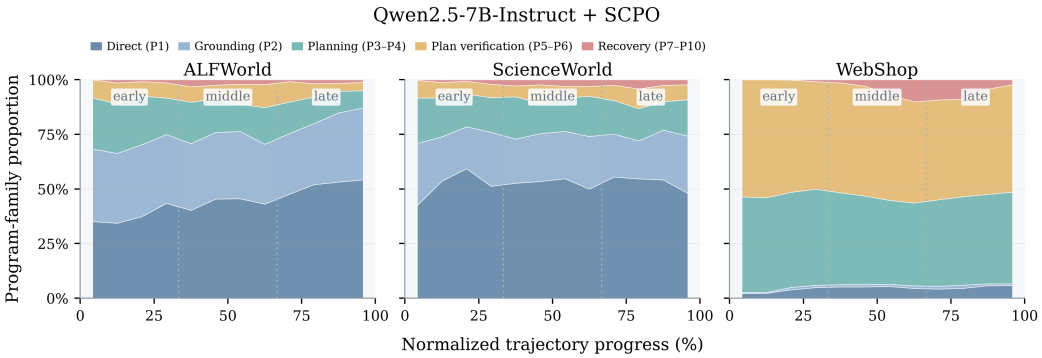

Figure 3: **Cognitive program usage over normalized trajectory progress for Qwen2.5-7B.** Programs are grouped into five mutually exclusive families:

Table 3: Ablation of SCPO on Qwen2.5-7B. CF denotes sparse counterfactual program credit and Cost denotes cost-aware trajectory utility.

Variant	ALFWorld		ScienceWorld		WebShop	
	SR↑	Tok.↓	SR↑	Tok.↓	SR↑	Tok.↓
SCPO (Full)	92.5	537.8	76.5	957.1	61.4	820.3
w/o CF Credit	83.5	692.4	65.0	716.2	55.6	1021.8
w/o Cost	87.5	1739.5	74.0	2167.8	58.2	1555.2

evolution procedure to CogRouter also yields positive improvements, demonstrating that test-time cognitive evolution can transfer across different cognitive representations. Additional results with Llama3.1-8B-Instruct are provided in Appendix A.4.

5 Analysis

5.1 What Cognitive Control Does SCPO Learn?

Although the main results show that SCPO improves task performance, they do not reveal how the Router allocates different cognitive programs during interaction. We therefore analyze how program usage varies over normalized trajectory progress and across environments. For visualization, we group the ten programs into five mutually exclusive families.

As shown in Figure 3, SCPO does not collapse to a single cognitive pattern. In ALFWorld, grounding-oriented programs are more prominent early in the trajectory, while direct execution becomes increasingly common toward later stages. ScienceWorld maintains a more heterogeneous mixture of direct, grounding, planning, and verification programs throughout the interaction. WebShop exhibits a different profile, with planning and plan-verification programs dominating most of the trajectory. These environment-specific patterns indicate that SCPO learns different cognitive allocations rather than a uniform preference for more complex or costly programs. Recovery programs are activated only in states with explicit transition failure and are therefore interpreted separately from the normal-state program families. Results for Llama3.1-8B show the same qualitative absence of program collapse and are provided in Appendix A.4.

5.2 What Drives SCPO?

To identify the source of SCPO's gains, we ablate its two key components: sparse counterfactual program credit and cost-aware trajectory utility. Table 3 shows that removing counterfactual comparisons consistently degrades task performance across all three environments, reducing success rate by 9.0, 11.5, and 5.8 percentage points on ALFWorld, ScienceWorld, and WebShop, respectively.

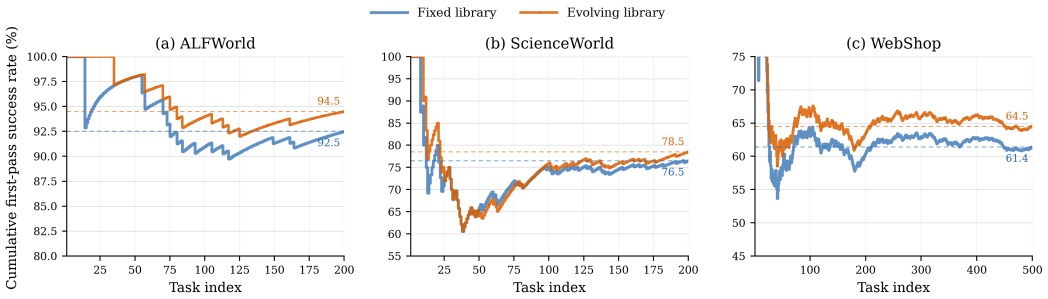

Figure 4: **Cumulative impact of test-time cognitive evolution.** On Qwen2.5-7B-Instruct, the evolving library incorporates only verified cognitive updates and improves first-pass performance over sequential tasks without parameter updates.

Since this variant retains the same Program-SFT initialization and ordinary on-policy objective, the result indicates that trajectory-level feedback is less effective at assigning credit to *which* cognitive program should be selected. State-matched counterfactual comparisons provide a more direct program-level learning signal.

Removing the cost-aware utility leads to a smaller but consistent performance drop, while substantially increasing token consumption across all three environments. In particular, token usage rises to approximately 1.9–$3.2\times$ that of full SCPO. These results reveal complementary roles of the two components: counterfactual credit primarily improves program selection, whereas cost-aware utility is critical for maintaining an effective performance–computation trade-off.

5.3 How Does Cognitive Evolution Modify Reasoning?

While Table 2 reports the overall gains after evolution, it does not reveal whether these improvements come from temporary retries or persistent cognitive updates. We therefore evaluate agents in a sequential task stream, where each accepted cognitive update can only influence future tasks.

Figure 4 compares a fixed cognitive library with an evolving library. Across ALFWorld, ScienceWorld, and WebShop, the evolving library achieves higher first-pass success as verified cognitive updates accumulate. Since each task is evaluated only with its first interaction attempt, these improvements cannot be attributed to additional retries. Instead, they reflect the reuse of previously acquired cognitive modifications.

These results demonstrate that test-time evolution transforms isolated failure experiences into persistent updates to the external cognitive library. Qualitative examples of evolved programs and library evolution dynamics are provided in Appendix A.5.

6 Conclusion

We introduced EvoCog, a framework that represents LLM agent cognition as composable programs and enables their optimization and evolution. Through program-conditioned SFT, Sparse Counterfactual Program Optimization, and verified test-time cognitive evolution, EvoCog learns to select, execute, and improve cognitive procedures without updating model parameters. Experiments on ALFWorld, ScienceWorld, and WebShop demonstrate that structured cognitive programs provide an effective basis for adaptive reasoning and experience-driven agent improvement.

A limitation of the current framework is that test-time evolution relies on an external diagnosis module to identify failure causes and propose residual modifications. Future work will explore more autonomous evolution mechanisms that allow agents to discover and verify cognitive improvements with reduced external supervision.

AI USE STATEMENT

In this work, we used generative AI tools for language polishing, proofreading, and limited assistance in literature organization during manuscript preparation. We have not used generative AI tools for developing the proposed method, designing experiments, generating experimental results, or drawing scientific conclusions. All research ideas, algorithmic designs, implementations, evaluations, and analyses were conducted and verified by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Anthropic. The Claude 4 Model Family and Sonnet 4.6 Technical Report. https://www.anthropic.com, 2026. Technical Report.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Mengyi Deng, Xin Li, Duyi Pan, Zilin Wang, Zhiwei Li, Zhijiang Guo, and Wei Wang. Reskill: Explicit failure attribution and structured repair for interactive language agents. *arXiv preprint arXiv:2609.15684*, 2026.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aileen Zhang, Akshat Bihani, Akshat Gupta, Akshila Singh, Aditya Sankar, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In *International Conference on Machine Learning*, 2025.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training, 2025.

Yao Fu et al. Autoguide: Automated generation and optimization of agent workflows, 2024. arXiv preprint.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Tianyi Jiang, Arctanx An, Hengyi Feng, et al. Chain of mindset: Reasoning with adaptive cognitive modes, 2026.

Jeonghye Kim, Sojeong Rhee, Minbeom Kim, Dohyung Kim, Sangmook Lee, Youngchul Sung, and Kyomin Jung. Reflact: World-grounded decision making in llm agents via goal-state reflection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

OpenAI. Hello gpt-4o. https://openai.com/index/hello-gpt-4o/, 2024.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li,

Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxian Song, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, Alex Makelov, Alex Neitz, Alex Wei, Alexandra Barr, Alexandre Kirchmeyer, Alexey Ivanov, Alexi Christakis, Alistair Gillespie, Allison Tam, Ally Bennett, Alvin Wan, Alyssa Huang, Amy McDonald Sandjideh, Amy Yang, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrei Gheorghe, Andres Garcia Garcia, Andrew Braunstein, Andrew Liu, Andrew Schmidt, Andrey Mereskin, Andrey Mishchenko, Andy Applebaum, Andy Rogerson, Ann Rajan, Annie Wei, Anoop Kotha, Anubha Srivastava, Anushree Agrawal, Arun Vijayvergiya, Ashley Tyra, Ashvin Nair, Avi Nayak, Ben Eggers, Bessie Ji, Beth Hoover, Bill Chen, Blair Chen, Boaz Barak, Borys Minaiev, Botao Hao, Bowen Baker, Brad Lightcap, Brandon McKinzie, Brandon Wang, Brendan Quinn, Brian Fioca, Brian Hsu, Brian Yang, Brian Yu, Brian Zhang, Brittany Brenner, Callie Riggins Zetino, Cameron Raymond, Camillo Lugaresi, Carolina Paz, Cary Hudson, Cedric Whitney, Chak Li, Charles Chen, Charlotte Cole, Chelsea Voss, Chen Ding, Chen Shen, Chengdu Huang, Chris Colby, Chris Hallacy, Chris Koch, Chris Lu, Christina Kaplan, Christina Kim, CJ Minott-Henriques, Cliff Frey, Cody Yu, Coley Czarnecki, Colin Reid, Colin Wei, Cory Decareaux, Cristina Scheau, Cyril Zhang, Cyrus Forbes, Da Tang, Dakota Goldberg, Dan Roberts, Dana Palmie, Daniel Kappler, Daniel Levine, Daniel Wright, Dave Leo, David Lin, David Robinson, Declan Grabb, Derek Chen, Derek Lim, Derek Salama, Dibya Bhattacharjee, Dimitris Tsipras, Dinghua Li, Dingli Yu, DJ Strouse, Drew Williams, Dylan Hunn, Ed Bayes, Edwin Arbus, Ekin Akyurek, Elaine Ya Le, Elana Widmann, Eli Yani, Elizabeth Proehl, Enis Sert, Enoch Cheung, Eri Schwartz, Eric Han, Eric Jiang, Eric Mitchell, Eric Sigler, Eric Wallace, Erik Ritter, Erin Kavanaugh, Evan Mays, Evgenii Nikishin, Fangyuan Li, Felipe Petroski Such, Filipe de Avila Belbute Peres, Filippo Raso, Florent Bekerman, Foivos Tsimpourlas, Fotis Chantzis, Francis Song, Francis Zhang, Gaby Raila, Garrett McGrath, Gary Briggs, Gary Yang, Giambattista Parascandolo, Gildas Chabot, Grace Kim, Grace Zhao, Gregory Valiant, Guillaume Leclerc, Hadi Salman, Hanson Wang, Hao Sheng, Haoming Jiang, Haoyu Wang, Haozhun Jin, Harshit Sikchi, Heather Schmidt, Henry Aspegren, Honglin Chen, Huida Qiu, Hunter Lightman, Ian Covert, Ian Kivlichan, Ian Silber, Ian Sohl, Ibrahim Hammoud, Ignasi Clavera, Ikai Lan, Ilge Akkaya, Ilya Kostrikov, Irina Kofman, Isak Etinger, Ishaan Singal, Jackie Hehir, Jacob Huh, Jacqueline Pan, Jake Wilczynski, Jakub Pachocki, James Lee, James Quinn, Jamie Kiros, Janvi Kalra, Jasmyn

Samaroo, Jason Wang, Jason Wolfe, Jay Chen, Jay Wang, Jean Harb, Jeffrey Han, Jeffrey Wang, Jennifer Zhao, Jeremy Chen, Jerene Yang, Jerry Tworek, Jesse Chand, Jessica Landon, Jessica Liang, Ji Lin, Jiancheng Liu, Jianfeng Wang, Jie Tang, Jihan Yin, Joanne Jang, Joel Morris, Joey Flynn, Johannes Ferstad, Johannes Heidecke, John Fishbein, John Hallman, Jonah Grant, Jonathan Chien, Jonathan Gordon, Jongsoo Park, Jordan Liss, Jos Kraaijeveld, Joseph Guay, Joseph Mo, Josh Lawson, Josh McGrath, Joshua Vendrow, Joy Jiao, Julian Lee, Julie Steele, Julie Wang, Junhua Mao, Kai Chen, Kai Hayashi, Kai Xiao, Kamyar Salahi, Kan Wu, Karan Sekhri, Karan Sharma, Karan Singhal, Karen Li, Kenny Nguyen, Keren Gu-Lemberg, Kevin King, Kevin Liu, Kevin Stone, Kevin Yu, Kristen Ying, Kristian Georgiev, Kristie Lim, Kushal Tirumala, Kyle Miller, Lama Ahmad, Larry Lv, Laura Clare, Laurance Fauconnet, Lauren Itow, Lauren Yang, Laurentia Romaniuk, Leah Anise, Lee Byron, Leher Pathak, Leon Maksin, Leyan Lo, Leyton Ho, Li Jing, Liang Wu, Liang Xiong, Lien Mamitsuka, Lin Yang, Lindsay McCallum, Lindsey Held, Liz Bourgeois, Logan Engstrom, Lorenz Kuhn, Louis Feuvrier, Lu Zhang, Lucas Switzer, Lukas Kondraciuk, Lukasz Kaiser, Manas Joglekar, Mandeep Singh, Mandip Shah, Manuka Stratta, Marcus Williams, Mark Chen, Mark Sun, Marselus Cayton, Martin Li, Marvin Zhang, Marwan Aljubeh, Matt Nichols, Matthew Haines, Max Schwarzer, Mayank Gupta, Meghan Shah, Melody Y. Guan, Melody Huang, Meng Dong, Mengqing Wang, Mia Glaese, Micah Carroll, Michael Lampe, Michael Malek, Michael Sharman, Michael Zhang, Michele Wang, Michelle Pokrass, Mihai Florian, Mikhail Pavlov, Miles Wang, Ming Chen, Mingxuan Wang, Minnia Feng, Mo Bavarian, Molly Lin, Moose Abdool, Mostafa Rohaninejad, Nacho Soto, Natalie Staudacher, Natan LaFontaine, Nathan Marwell, Nelson Liu, Nick Preston, Nick Turley, Nicklas Ansman, Nicole Blades, Nikil Pancha, Nikita Mikhaylin, Niko Felix, Nikunj Handa, Nishant Rai, Nitish Keskar, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Oona Gleeson, Pamela Mishkin, Patryk Lesiewicz, Paul Baltescu, Pavel Belov, Peter Zhokhov, Philip Pronin, Phillip Guo, Phoebe Thacker, Qi Liu, Qiming Yuan, Qinghua Liu, Rachel Dias, Rachel Puckett, Rahul Arora, Ravi Teja Mullapudi, Raz Gaon, Reah Miyara, Rennie Song, Rishabh Aggarwal, RJ Marsan, Robel Yemiru, Robert Xiong, Rohan Kshirsagar, Rohan Nuttall, Roman Tsiupa, Ronen Eldan, Rose Wang, Roshan James, Roy Ziv, Rui Shu, Ruslan Nigmatullin, Saachi Jain, Saam Talaie, Sam Altman, Sam Arnesen, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Sarah Yoo, Savannah Heon, Scott Ethersmith, Sean Grove, Sean Taylor, Sebastien Bubeck, Sever Banesiu, Shaokyi Amdo, Shengjia Zhao, Sherwin Wu, Shibani Santurkar, Shiyu Zhao, Shraman Ray Chaudhuri, Shreyas Krishnaswamy, Shuaiqi, Xia, Shuyang Cheng, Shyamal Anadkat, Simón Posada Fishman, Simon Tobin, Siyuan Fu, Somay Jain, Song Mei, Sonya Egoian, Spencer Kim, Spug Golden, SQ Mah, Steph Lin, Stephen Imm, Steve Sharpe, Steve Yadlowsky, Sulman Choudhry, Sungwon Eum, Suvansh Sanjeev, Tabarak Khan, Tal Stramer, Tao Wang, Tao Xin, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Degry, Thomas Shadwell, Tianfu Fu, Tianshi Gao, Timur Garipov, Tina Sriskandarajah, Toki Sherbakov, Tomek Korbak, Tomer Kaftan, Tomo Hiratsuka, Tongzhou Wang, Tony Song, Tony Zhao, Troy Peterson, Val Kharitonov, Victoria Chernova, Vineet Kosaraju, Vishal Kuo, Vitchyr Pong, Vivek Verma, Vlad Petrov, Wanning Jiang, Weixing Zhang, Wenda Zhou, Wenlei Xie, Wenting Zhan, Wes McCabe, Will DePue, Will Ellsworth, Wulfie Bain, Wyatt Thompson, Xiangning Chen, Xiangyu Qi, Xin Xiang, Xinwei Shi, Yann Dubois, Yaodong Yu, Yara Khakbaz, Yifan Wu, Yilei Qian, Yin Tat Lee, Yinbo Chen, Yizhen Zhang, Yizhong Xiong, Yonglong Tian, Young Cha, Yu Bai, Yu Yang, Yuan Yuan, Yuanzhi Li, Yufeng Zhang, Yuguang Yang, Yujia Jin, Yun Jiang, Yunyun Wang, Yushi Wang, Yutian Liu, Zach Stubenvoll, Zehao Dou, Zheng Wu, and Zhigang Wang. Openai gpt-5 system card, 2026. URL https://arxiv.org/abs/2601.03267.

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.

X Wang et al. Awm: Agent workflow memory, 2025. arXiv preprint.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Z Xi, Y Ding, W Chen, B Hong, H Guo, J Wang, D Yang, C Liao, X Guo, W He, et al. Agentgym: Evolving large language model-based agents across diverse environments. arxiv 2024. *arXiv preprint arXiv:2406.04151*.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. Ares: Adaptive reasoning effort selection for efficient llm agents, 2026a.

Ruihan Yang, Fanghua Ye, et al. Think fast and slow: Step-level cognitive depth adaptation for llm agents, 2026b.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Jiajie Zhang, Nianyi Lin, and Juanzi Li. Adaptthink: Reasoning models can learn when to think. *arXiv preprint arXiv:2505.13417*, 2025.

Andrew Zhao et al. Expel: A systematic framework for memory-enhanced exploration in large language model agents, 2024. arXiv preprint.

A Appendix

A.1 Benchmark Details

We evaluate EvoCog on three interactive benchmarks that cover embodied manipulation, scientific reasoning, and web-based decision making.

ALFWorld. ALFWorld (Shridhar et al., 2021) is a text-based embodied environment aligned with household manipulation tasks from the ALFRED benchmark. Agents interact with simulated household worlds through language actions and must complete long-horizon tasks involving navigation, object manipulation, and state tracking. The benchmark contains task types including Pick & Place,Examine in Light, Clean & Place, Heat & Place, Cool & Place and Pick Two & Place. These tasks require agents to ground textual observations, maintain intermediate states, and execute multi-step action sequences. We report task success rate following the standard evaluation protocol. The maximum interaction steps is set to 30 for evaluation.

ScienceWorld. ScienceWorld (Wang et al., 2022) is an interactive text environment designed to evaluate grounded scientific reasoning. Agents must complete scientific tasks by interacting with a simulated laboratory environment, including performing experiments, manipulating objects, making observations, and following scientific procedures. The benchmark covers diverse elementary science scenarios, such as changes of matter states, chemical reactions, temperature-related experiments, and biological processes. Successful completion requires not only retrieving scientific knowledge but also planning and executing valid action sequences under environmental constraints. We report both environment score and success rate. The maximum interaction steps is set to 100 for evaluation.

WebShop. WebShop (Yao et al., 2022) is a simulated e-commerce environment for evaluating web interaction and decision-making capabilities. Given a natural language shopping instruction, agents must search through product pages, inspect product attributes, navigate available options, and select an item that satisfies user constraints. Tasks require instruction understanding, information retrieval, constraint satisfaction, and sequential decision making over a large product space. We report the standard WebShop score and success rate. The maximum interaction steps is set to 15 for evaluation.

A.2 Implementation Details

A.2.1 Training Dataset Construction

We construct SFT and SCPO RL data from expert trajectories in AgentTraj-L (Xi et al.). For program-conditioned SFT, we randomly sample 500 expert trajectories from each environment,

including ALFWorld, ScienceWorld, and WebShop. Each expert trajectory is expanded into individual decision states. For each state, a grammar-valid cognitive program is sampled from the legal program set with balanced coverage over the initial program library. We then prompt GPT-4o to generate the corresponding structured cognitive trace following the assigned program, while preserving the expert environment action as the final action target. An example of the sft data format is shown as follow.

```
prompt:You are an executor operating in the alfworld environment.  The
cognitive program below was assigned externally.  It is not a label to
predict and you must not choose, reorder, add, or omit operations.

Task:
put some creditcard on armchair.

Previous transition history:  (no previous transitions)

Current step:  1
Current observation:
You are in the middle of a room.  Looking quickly around you, you see
a armchair 1, a coffeetable 1, a garbagecan 1, a sidetable 1, a sofa
1, and a tvstand 1.

Your task is to:  put some creditcard on armchair.

Current admissible actions:
- go to armchair 1
- go to coffeetable 1
- go to garbagecan 1
- go to sidetable 1
- go to sofa 1
- go to tvstand 1
- inventory
- look

Assigned cognitive program:
<program>GROUND -> PLAN -> ACT</program>

Operation contract:
- GROUND: extract task-relevant facts.  Do not plan.
- PLAN: state a short-term subgoal.  Do not emit an action.
- ACT: emit exactly one environment action.

ALFWorld action syntax:
go to [recep id]; take [obj id] from [recep id];
put [obj id] in/on [recep id]; open/close [recep id].

Execute the assigned program in order.

Output:
<program>GROUND -> PLAN -> ACT</program>
<ground>The task requires placing a credit card on the armchair.The
room contains an armchair, a sofa, and other furniture.</ground>
<plan>Identify and locate a credit card in the room, then move towards
it to pick it up.</plan>
<action>go to sofa 1</action>
```

For SCPO training, we construct a separate task-level dataset from the remaining training tasks after excluding all environments used for SFT. Unlike SFT, RL data does not contain expert actions or predefined cognitive traces; the current policy generates both program selections and interaction

Table 4: Initial cognitive program library. C_p denotes CHECK(PLAN) and C_t denotes CHECK(LAST TRANSITION).

Program	Context	Cognitive procedure
P1	Normal	ACT
P2	Normal	GROUND → ACT
P3	Normal	PLAN → ACT
P4	Normal	GROUND → PLAN → ACT
P5	Normal	PLAN → CHECK(plan) → ACT
P6	Normal	GROUND → PLAN → CHECK(plan) → ACT
P7	Recovery	CHECK(last_transition) → ACT
P8	Recovery	CHECK(last_transition) → PLAN → ACT
P9	Recovery	GROUND → CHECK(last_transition) → ACT
P10	Recovery	GROUND → CHECK(last_transition) → PLAN → ACT

trajectories online. The resulting datasets contain 1,920 ALFWorld tasks, 1,620 ScienceWorld tasks, and 5,705 WebShop tasks after removing overlap with the SFT data.

A.2.2 PROGRAM LIBRARY AND COGNITIVE PROMPT

EvoCog represents cognition through four typed atomic operations: GROUND, PLAN, CHECK, and ACT. Each operation follows a strict execution contract to ensure that different cognitive procedures remain compositional and that individual operations do not implicitly perform the role of other components.

```
Ground(G): Extract only task-relevant facts from the current state
and history. Do not plan.
Plan(P): State a short-term subgoal and a small set of candidate
steps. Do not emit an environment action.
Check(plan)(C_p): Verify the immediately preceding PLAN against state
facts and action constraints. Do not replace it with a new plan.
Check(last_transition)(C_t): diagnose the explicitly failed previous
transition from its action and feedback. Do not claim a failure not
present in the feedback.
Act(A): Emit exactly one environment action and nothing else inside
<action>.
```

These atomic operations are composed under the causal grammar. The initial cognitive library contains ten grammar-valid programs covering normal interaction and recovery scenarios. Table 4 lists the complete program set used during SFT and RL.

A.2.3 SCPO IMPLEMENTATION DETAILS

Online Rollout and Advantage Normalization. After program-conditioned SFT initialization, SCPO performs online interaction with the environment using the current Router and Executor. At each optimization iteration, we sample 8 tasks and collect 8 trajectories for each task. The collected trajectories are used for both ordinary policy optimization and sparse counterfactual program comparison.

For each trajectory τ, SCPO computes the cost-aware utility:

$$U(\tau) = R_{\text{task}}(\tau) - \lambda C(\tau),$$

where R_{task} denotes task-level performance and $C(\tau)$ includes reasoning-token usage, interaction steps, invalid executions, and formatting violations.

For ordinary rollouts generated from the same task, utilities are normalized to obtain the trajectory advantage:

$$A_i^{\text{base}} = \frac{U_i - \mu_{\mathcal{G}}}{\sigma_{\mathcal{G}} + \epsilon},$$

where $\mu_{\mathcal{G}}$ and $\sigma_{\mathcal{G}}$ are the mean and standard deviation of utilities within the rollout group $\mathcal{G}$. The normalized advantage is shared by the Executor and Router objectives.

Sparse Anchor Selection. Exhaustively evaluating all programs at every interaction step would require a large number of additional environment rollouts. SCPO therefore performs counterfactual evaluation only at sparse anchor states where program selection is likely to influence downstream outcomes.

An interaction state is considered an anchor candidate if it satisfies one of the following conditions: (1) invalid execution or explicit environment failure feedback is observed; (2) the trajectory shows repeated unsuccessful progress or stagnation; or (3) the state corresponds to a critical decision boundary in an unsuccessful trajectory. From the candidate states, SCPO selects a small number of anchors for counterfactual comparison.

Executor and Router Optimization. Ordinary trajectories provide PPO-style learning signals for the Executor and Router:

$$\mathcal{L}_{\text{exec}} = \mathcal{L}_{\text{PPO}}(\pi_\theta, A^{\text{base}}),$$

$$\mathcal{L}_{\text{router}} = \mathcal{L}_{\text{PPO}}(\pi_\theta, A^{\text{base}}).$$

The Executor learns how to realize the selected cognitive program, while the Router learns state-dependent program selection.

Counterfactual Preference Optimization. For each incumbent–challenger pair, SCPO introduces an additional preference objective for Router optimization. Let P_a and P_b denote the two compared programs. Their utility difference is:

$$\Delta_{ab} = \hat{U}(s, P_a) - \hat{U}(s, P_b).$$

The relative Router preference is measured by:

$$d_{ab} = [\log \pi_\theta(P_a|s) - \log \pi_\theta(P_b|s)] - [\log \pi_{\text{ref}}(P_a|s) - \log \pi_{\text{ref}}(P_b|s)] \,.$$

The utility-weighted preference loss is:

$$\mathcal{L}_{\text{cf}} = |\Delta_{ab}| \,\text{softplus}\left(-\beta \,\text{sign}(\Delta_{ab}) d_{ab}\right),$$

where β controls the preference sharpness. Pairs with negligible utility differences are ignored.

Complete SCPO Objective. The final optimization objective combines ordinary policy optimization and counterfactual program preference:

$$\mathcal{L}_{\text{SCPO}} = \mathcal{L}_{\text{exec}} + \alpha \mathcal{L}_{\text{router}} + \lambda_{\text{cf}} \mathcal{L}_{\text{cf}} - \lambda_H H_R + \lambda_{KL} D_{KL}.$$

The entropy term prevents premature Router collapse, while the KL regularizer constrains deviation from the SFT initialization. The algorithm is shown in Algorithm 1

Algorithm 1 Sparse Counterfactual Program Optimization (SCPO)

Require: Program-conditioned SFT policy π_θ, reference policy π_{ref}, cognitive program library $\mathcal{L}$

for each optimization iteration **do**
  Sample a batch of tasks from the training environments
  Collect online trajectories using the current Router and Executor
  for each trajectory group **do**
    Compute trajectory utilities $U(\tau)$ and normalized advantages A^{base}
    Identify sparse anchor states using failure-sensitive criteria
    for each selected anchor state s **do**
      Record incumbent program P_{inc} selected by the Router
      Sample one legal challenger program $P_{\text{chal}} \in \mathcal{S}(s) \setminus \{P_{\text{inc}}\}$
      Restore the environment state and execute the challenger suffix rollout
      Compute counterfactual utility difference $\Delta(s)$
    end for
    Compute Executor PPO loss, Router PPO loss, and counterfactual preference loss
  end for
  Update π_θ using $\mathcal{L}_{\text{SCPO}}$
end for

Ensure: Optimized Router and Executor π_θ

A.3 Training Details

Hyperparameters Table 5 summarizes the training configurations of all experiments. Unless otherwise specified, the same training pipeline and optimization settings are applied across ALFWorld, ScienceWorld, and WebShop. Both supervised fine-tuning and reinforcement learning stages are implemented with the verl framework (Sheng et al., 2025).

Training Cost. We further report the training cost of each stage in Table 6. The reported time includes data preparation, model optimization, and online environment interaction required by each stage.

A.4 Extended Results

Results on Llama3.1-8B-Instruct. We further evaluate EvoCog using Llama3.1-8B-Instruct as the backbone. The results are summarized in Table 7. EvoCog maintains consistent improvements over reinforcement-learning and adaptive-reasoning baselines across all evaluated environments.

Table 8 further reports test-time cognitive evolution results. Similar to Qwen2.5-7B-Instruct, EvoCog obtains additional improvements after verified cognitive updates.

Additional Cognitive Control Analysis. Following Section 5.1, we analyze cognitive program allocation of Llama3.1-8B-Instruct. The Router exhibits similar environment-dependent program usage patterns without collapsing into a single cognitive strategy.

Additional Evolution Dynamics. We further visualize cumulative first-pass success during sequential evaluation. The evolving library improves future task performance compared with the fixed library while keeping model parameters unchanged.

A.5 Case Study

Figure 7 presents an example of how EvoCog converts a failed interaction into a reusable cognitive modification. The agent first fails due to repeated invalid action execution. The diagnosis module attributes the failure to missing action verification, after which a guarded residual is attached to the original cognitive program. The verified variant enables successful completion while keeping the model parameters and Router program space unchanged.

Table 5: Training hyperparameters of EvoCog on Qwen2.5-7B-Instruct and Llama3.1-8B-Instruct.

Stage	Hyperparameter	Qwen2.5-7B	Llama3.1-8B
Program-SFT	Learning rate	2×10^{-6}	2×10^{-6}
	Batch size	32	32
	Epochs	3	3
	Max sequence length	4096	4096
	Optimizer	AdamW	AdamW
GRPO	Learning rate	5×10^{-7}	5×10^{-7}
	Max prompt length	4096	4096
	Max response length	1024	1024
	Group size	8	8
	Rollouts per task	8	8
	Epochs	1	1
	KL Coefficient	0.01	0.01
	Clip ratio	0.2	0.2
	Rollout temperature	1.0	1.0
	Validation temperature	0.0	0.0
	Mini-batch	8	8
GIGPO	Learning rate	5×10^{-7}	5×10^{-7}
	Max prompt length	4096	4096
	Max response length	1024	1024
	Group size	8	8
	Rollouts per task	8	8
	Epochs	1	1
	KL Coefficient	0.01	0.01
	Clip ratio	0.2	0.2
	Rollout temperature	1.0	1.0
	Validation temperature	0.0	0.0
	Mini-batch	8	8
SCPO	Learning rate	5×10^{-7}	5×10^{-7}
	Max prompt length	4096	4096
	Max response length	1024	1024
	Group size	8	8
	Rollouts per task	8	8
	Epochs	1	1
	KL Coefficient	0.01	0.01
	Clip ratio	0.2	0.2
	Rollout temperature	1.0	1.0
	Validation temperature	0.0	0.0
	Mini-batch	8	8
	α	0.1	0.1
	λ_{cf}	0.5	0.5

A.6 BASELINE

Prompting-based Agents. We evaluate several inference-time agent baselines that improve decision making through prompting or external reasoning procedures. Direct prompting generates environment actions without explicit reasoning steps. ReAct (Yao et al., 2023) interleaves reasoning traces with environment actions, and Reflexion (Shinn et al., 2023) augments the agent with verbal reflection after failed trials. Plan-and- Act (Erdogan et al., 2025) explicitly separates planning and execution, while ReflAct (Kim et al., 2025) incorporates goal-state reflection to improve long-horizon decision making.

For frontier language models, including GPT-5 (Singh et al., 2026), Gemini 2.5 Pro (Comanici et al., 2025), Claude Sonnet 4.6 (Anthropic, 2026), and DeepSeek V3.2 (Liu et al., 2025), we use the same ReAct-style interaction protocol, where the model generates intermediate reasoning followed by environment actions. All prompting-based methods use the same task instructions, environment interfaces, and evaluation protocols as EvoCog.

Table 6: Training time and performance comparison of different methods on ALFWorld, ScienceWorld and WebShop.

Method	ALFWorld		ScienceWorld		WebShop	
	Training time	Performance	Training time	Performance	Training time	Performance
Qwen2.5-7B-Instruct						
GRPO	27h 18m	82.5	31h 2m	49.0	50h 45m	55.6
GiGPO	42h 16m	91.5	76h 36m	56.5	78h 03m	62
SCPO	36h 27m	92.5	48h 21m	76.5	70h 35m	61.4
Llama3.1-8B-Instruct						
GRPO	23h 22m	82.0	31h 55m	45.0	48h 06m	56.8
GiGPO	36h 44m	88.5	58h 04m	64.0	69h 42m	61.7
SCPO	32h 36m	93.0	35h 41m	77.0	53h 30m	64.0

Table 7: Main results on ALFWorld, ScienceWorld, and WebShop. SR denotes success rate (%) and Tok. denotes generated tokens. Best results within each backbone are highlighted.

Method	ALFWorld		ScienceWorld			WebShop			Average	
	SR↑	Tok.↓	Score↑	SR↑	Tok.↓	Score↑	SR↑	Tok.↓	SR↑	Tok.↓
Llama3.1-8B-Instruct										
Direct	21.5	253.0	29.9	14.5	310.0	62.3	30.6	312.0	22.20	291.7
ReAct	22.0	879.1	13.7	3.0	1268.8	47.6	27.4	1312.0	17.47	1153.3
Reflexion	18.5	3654.8	17.8	2.5	5072.4	63.3	32.8	2738.5	17.93	3821.9
Plan-and-Act	9.5	329.4	29.14	2.5	625.4	17.2	7.5	257.8	6.50	404.2
ReflAct	60.5	1240.1	47.2	33.0	1137.5	20.1	8.1	1024.7	33.87	1134.1
AdaptThink	78.5	1587.2	52.3	45.5	1587.8	56.3	38.5	1526.4	54.17	1567.1
CogRouter	91.5	811.6	83.7	70.5	963.8	74.7	60.5	825.6	74.17	867
Program-SFT	54	1429.9	31.6	23.5	1825.7	62.9	56.6	544.1	44.7	1266.56
+ GRPO	82.0	1174.2	65.7	45.0	948.3	72.1	56.8	623.5	61.27	915.3
+ GiGPO	88.5	898.7	81.1	64.0	1526.7	79.4	61.7	518.0	71.40	981.1
+ SCPO (Ours)	**93.0**	673.1	**87.9**	**77.0**	1200.2	**82.9**	**64**	572.2	**78.67**	815.2

Adaptive Reasoning Agents. We compare EvoCog with adaptive reasoning approaches that dynamically allocate reasoning computation during interaction. AdaptThink (Zhang et al., 2025) learns to adaptively select between thinking and non-thinking modes. CogRouter (Yang et al., 2026b) learns step-level routing among predefined cognitive levels with supervised initialization followed by reinforcement learning.

For CogRouter, we use the official cognitive-level formulation while adapting the environment interface to ALFWorld, ScienceWorld, and WebShop. In the test-time evolution experiments, we apply the same evolution procedure to the CogRouter to examine whether cognitive evolution transfers across different cognitive representations.

Table 8: Test-time cognitive evolution on Llama3.1-8b-Instruct.

Method	ALFWorld		ScienceWorld		WebShop	
	Gain	**SR**	**Gain**	**SR**	**Gain**	**SR**
Llama3.1-8B-Instruct						
SCPO+evo	+0.5	93.5	+2.0	79.0	+2.0	66.5
CogRouter+evo	+0.5	92.0	+5.0	75.5	+3.3	63.8

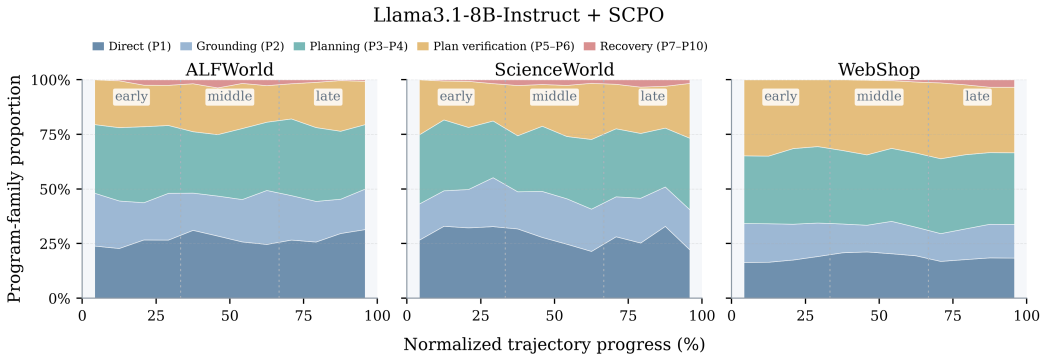

Figure 5: **Cognitive program usage over normalized trajectory progress for Llama3.1-8b.** Programs are grouped into five mutually exclusive families.

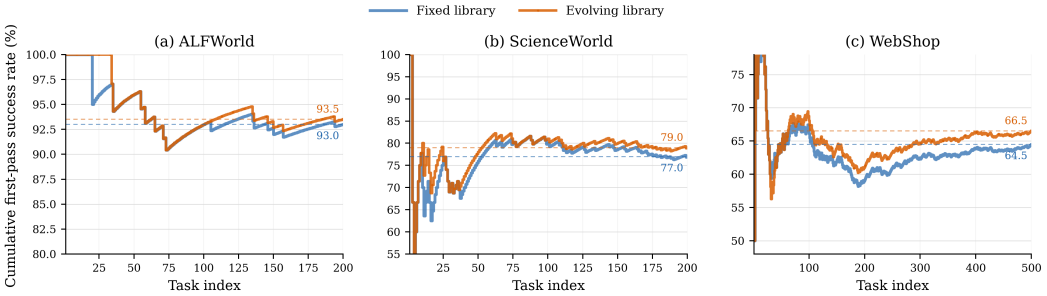

Figure 6: **Cumulative impact of test-time cognitive evolution.** On Llama3.1-8B-Instruct, the evolving library incorporates only verified cognitive updates and improves first-pass performance over sequential tasks without parameter updates.

Reinforcement Learning Agents. We compare SCPO with GRPO (Shao et al., 2024) and GiGPO (Feng et al., 2025), which optimize agent behavior through online policy optimization with different credit assignment strategies.

To provide a controlled comparison, both GRPO and GiGPO are initialized from the same program-conditioned SFT checkpoint as EvoCog and trained with the same training environments, task splits, rollout budgets, and backbone models.

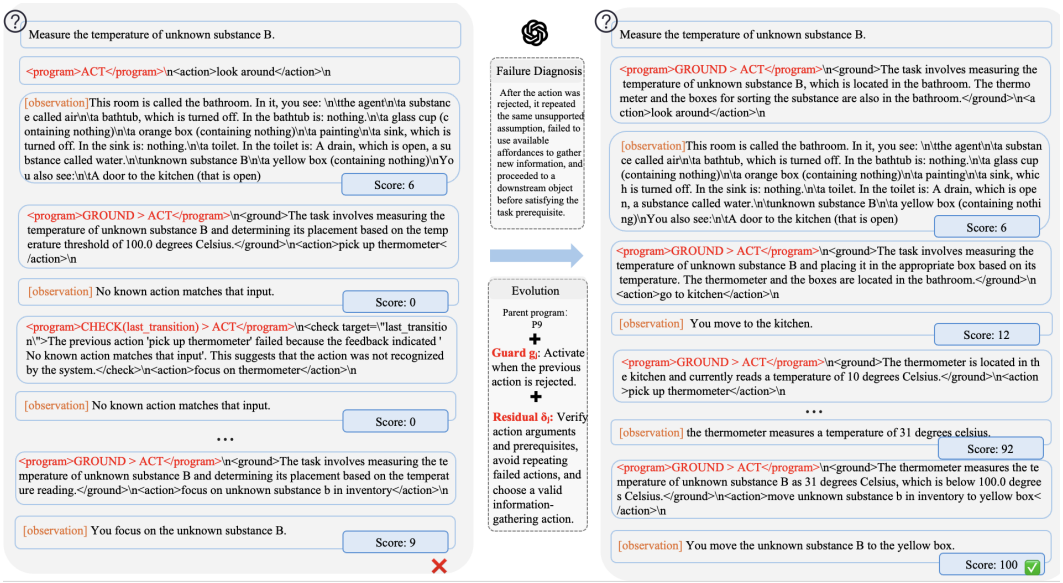

Figure 7: **Qualitative example of test-time cognitive evolution in ScienceWorld.** The left panel shows a failed trajectory before evolution, where repeated execution errors prevent task completion. The diagnosis module identifies the missing cognitive condition and generates a guarded residual attached to the original program. After verification, the evolved cognitive variant enables the agent to complete the task successfully without updating model parameters.