TRAVEL BACK TO MOVE FORWARD: RETROSPECTIVE EXPLORATION AND EVOLVING FOR LONG HORIZON AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents increasingly operate in long-horizon environments where early mistakes can trigger cascading failures and trap agents in dead-end trajectories. Agents must therefore be able to proactively recover from such states and turn their exploratory experience into guidance for subsequent decisions. We introduce **ReTRAIL**, a framework that enables agents to rewind the environment to earlier states while preserving valid progress and carrying forward knowledge learned from abandoned exploration. The environment state moves backward while the agent's knowledge continues to move forward. ReTRAIL trains a shared policy with two complementary signals: State-Anchored GRPO compares alternative continuations from revisited states to improve execution, while a branch-advancement reward guides reflection through the progress of subsequent exploration. Experiments across four interactive benchmark MineSweeper, Sokoban, ALFWorld and ScienceWorld show that simply adding a rewind primitive provides limited benefit and can even degrade performance. In contrast, ReTRAIL enables agents to rewind from unproductive states and use insights from abandoned branches to guide subsequent exploration. These results demonstrate the value of coupling environment rewinding with experience reuse within an ongoing task for long-horizon task solving.

1 INTRODUCTION

Large language model (LLM) agents are rapidly evolving from conversational systems into autonomous decision-making agents capable of solving long-horizon tasks through extended interactions, diverse tool use, and complex state transitions (Jimenez et al., 2024; Sun et al., 2026b; He et al., 2026b). As task complexity increases, agents inevitably encounter unfamiliar constraints and make early mistakes that may alter the environment, trigger cascading failures, and accumulate irrelevant information into the context (Zhou et al., 2024; Ye et al., 2026). Recovering from such failures requires agents to revisit earlier viable environment states without discarding either valid progress or useful experience. Crucially, returning to an earlier state should not mean returning to the same decisions: agents must distill what they have learned from abandoned exploration and use it to guide a more informed continuation.

Prior work has sought to guide LLMs to recover from errors and learn from unsuccessful history through reflection, repeated trials, and backtracking. Interleaving reasoning with acting lets an agent revise its plan at each step, but the revision stays local and the execution keeps moving forward (Yao et al., 2023; Choi et al., 2026). Cross-episode adaptation uses accumulated experience to guide new attempts (Shinn et al., 2023; Ouyang et al., 2026; Jiang et al., 2026), but applies it after an episode terminates and execution restarts from the initial state, requiring earlier progress to be repeated. Backtracking and trajectory-revision methods enable agents to reconsider earlier decisions through tree search, prompted controllers, or supervised correction traces (Yuan et al., 2025; Li et al., 2025; Zhang et al., 2026). A central challenge is to train agents to proactively recover within an ongoing task and use the experience gained from abandoned branches to improve subsequent exploration. This calls for learning signals that improve both execution after a rewind and the experience carried forward.

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.

Our key observation is that environment states can be rewound while the knowledge gained from exploration should be accumulated. An unsuccessful branch may reveal hidden constraints, ineffective actions, or promising alternatives, allowing the agent to revisit an earlier state with a better-informed context. We introduce **ReTRAIL**, a framework that exposes an environment-level *rewind* interface to the agent, allowing it to restore earlier environment states. Building on this interface, we propose **Retrospective Exploration**: the agent proactively leaves an unproductive branch, rewinds the environment to an earlier state, distills useful information from the abandoned branch into a compact reflection, and resumes execution with the preserved execution prefix and accumulated experience. To jointly develop the agent's ability to rewind and distill useful experience from abandoned branches, we use **State-Anchored GRPO** to improve execution by comparing alternative continuations from the same environment state, and a **branch-advancement reward** to train reflection based on the relative progress of subsequent exploration.

We evaluate ReTRAIL across four diverse interactive benchmarks: MineSweeper (Li et al., 2024), Sokoban (Racanière et al., 2017), ALFWorld (Shridhar et al., 2021), and ScienceWorld (Wang et al., 2022). ReTRAIL achieves the highest success rates among the evaluated methods when rewind is enabled. Our experiments reveal that simply providing existing agents with a rewind operation does not consistently improve performance and can even degrade it. In contrast, ReTRAIL benefits from proactive rewinding across all four benchmarks and achieves further gains through reflection-guided retries. On MineSweeper, success increases from 58.8% without rewind to 63.5% with proactive rewinding, and reaches 95.3% with up to three reflection-guided attempts, compared with 70.8% for the strongest baseline under the same evaluation protocol. Further analysis shows that retry gains are larger on longer tasks and when abandoned branches reveal information useful for subsequent decisions. These findings highlight the value of learning to rewind and carry forward useful experience, enabling unsuccessful exploration to inform continued progress within an ongoing long-horizon task.

Our main contributions are summarized as follows:

- We introduce **ReTRAIL**, a framework for **Retrospective Exploration** through an environment-level rewind interface. Agents proactively revisit earlier states while preserving valid progress and carrying forward experience distilled from abandoned branches to guide subsequent exploration.
- We develop two complementary learning signals for a shared execution and reflection policy. **State-Anchored GRPO** compares alternative continuations from the same environment state to improve execution, while a **branch-advancement reward** trains reflection based on the relative progress of subsequent exploration.
- We evaluate ReTRAIL across four interactive benchmarks, demonstrating improved success through proactive rewinding and reflection-guided retries. Further experiments show larger gains on longer and harder tasks, highlighting the value of retaining experience to guide subsequent exploration.

2 Related Work

2.1 Long-Horizon Execution and Recovery

Sustaining reliable execution over long interaction sequences motivates research on planning, context management, and error recovery in LLM agents (Yao et al., 2023; Zhang et al., 2023; Yuan et al., 2025). Search-based methods explore alternative action sequences within context (Zhou et al., 2024; Choi et al., 2026). Agent-R learns decision correction from revision traces (Yuan et al., 2025). However, revising decisions through context alone does not reverse changes to the environment, which can leave agents trapped in unproductive states. WebRollback uses action, critique, and rollback modules to select recovery points and carry forward experience summaries, with prompting and supervised fine-tuning from demonstrations (Zhang et al., 2026). GA-Rollback uses a prompted assistant to detect and localize errors during inference, restores earlier environment states through reset-and-replay, and retains error feedback to guide subsequent actions (Li et al., 2025). ReTRAIL take a step forward to train a shared execution and reflection policy that couples environment-level rewind with reinforcement learning, enabling more diverse exploration and more effective interaction budgets using.

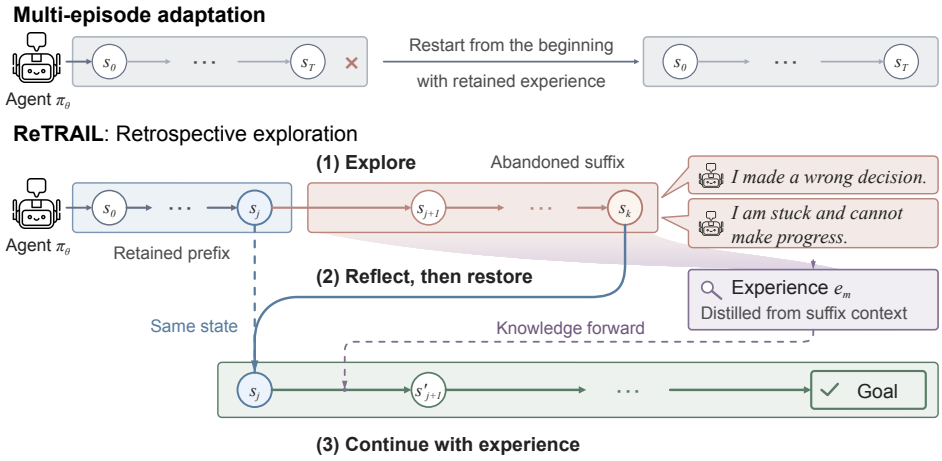

Figure 1: **Retrospective exploration with retained experience.** Multi-episode adaptation (top) restarts the task from s_0 after an unsuccessful attempt, carrying retained experience into a new episode. In contrast, ReTRAIL (bottom) proactively abandons an unproductive branch, distills its experience, rewinds to an earlier state, and resumes with the retained execution prefix and accumulated experience. Recovery can occur before terminal failure.

2.2 Test-Time Adaptation and Self-Evolving Agents

Adapting from interaction experience enables LLM agents to improve their behavior in unfamiliar tasks and environments. Reflection and memory mechanisms distill past trajectories into feedback and reusable strategies that guide subsequent decisions (Shinn et al., 2023; Ouyang et al., 2026). Beyond memory accumulation, EvoTest and Meta-TTL adapt prompts, memory, and tool-use routines from episode feedback (He et al., 2026c; Lou et al., 2026). Recent work further uses reinforcement learning to optimize how agents explore and adapt from experience. LaMer and MR-Search train exploration and post-episode reflection through meta-reinforcement learning, using discounted outcome reward to provide feedback for adaptation (Jiang et al., 2026; Xiao et al., 2026). Learning to Self-Evolve rewards context revisions according to downstream performance improvements (Chen et al., 2026b). ReTRAIL brings experience-informed adaptation into recovery within an ongoing task. By revisiting earlier environment states while preserving valid progress and distilled branch experience, agents can revise their exploration before the current attempt terminates, with subsequent branch progress providing a learning signal for reflection.

3 Retrospective Exploration

We formalize **Retrospective Exploration**, in which an agent rewinds to an earlier environment state while preserving the execution prefix and retaining experience from abandoned branches. We first introduce agent interaction and multi-episode adaptation, then describe how an environment-level *rewind* interface enables experience-informed recovery within an ongoing task.

Agent Interaction and Multi-Episode Adaptation. Given a task u, an LLM agent starts from environment state s_0. At step t, it observes o_t and selects an action $a_t \in \mathcal{A}$ based on the interaction history $h_t = (o_0, a_0, \ldots, a_{t-1}, o_t)$. The environment transitions to s_{t+1}, returning reward r_t and observation o_{t+1}. An episode yields a trajectory $\tau = (s_0, a_0, r_0, \ldots, a_{T-1}, r_{T-1}, s_T)$, ending upon task completion, environment termination, or exhaustion of its interaction budget.

Multi-episode adaptation groups successive attempts into a trial $\mathcal{T} = (\tau^{(0)}, \ldots, \tau^{(N-1)})$ (Jiang et al., 2026). Each episode restarts from s_0, while the policy conditions on experience $\mathcal{E}^{(n)}$ retained from preceding episodes:

$$a_t^{(n)} \sim \pi_\theta\big(\cdot \mid u, h_t^{(n)}, \mathcal{E}^{(n)}\big), \qquad \mathcal{E}^{(0)} = \varnothing. \tag{1}$$

The trial ends upon success or exhaustion of the attempt budget. Experience thus accumulates across episodes, but each restart requires rebuilding previously achieved environment progress.

Retrospective Exploration. ReTRAIL provides an environment-level *rewind* interface that allows the agent to restore a previously visited state during an ongoing attempt. When the current branch is unlikely to yield further progress, the agent can proactively rewind, preserve the execution prefix, and use experience distilled from the abandoned branch to guide a new continuation. We call this process *retrospective exploration*.

Execute. A forward segment σ_m is the contiguous interaction sequence from the initial state or a restored state to the next recovery boundary or trial termination. Rewinding ends the current branch and starts a new segment from the selected earlier state, while accumulated experience remains available. We index these segments by m and condition execution on the interaction history h_t and accumulated branch experience $\mathcal{E}_m = (e_0, \ldots, e_{m-1})$, where e_i is the reflection distilled at recovery i and $\mathcal{E}_0 = \varnothing$:

$$a_t \sim \pi_\theta(\cdot \mid u, h_t, \mathcal{E}_m). \tag{2}$$

Detect recovery. At s_{k_m}, recovery may be initiated proactively once the agent judges that the current branch is unlikely to yield further progress, or triggered by a failure signal. The agent selects an earlier state s_{j_m}, partitioning the current execution path into a retained prefix and an abandoned suffix:

$$\begin{aligned} \rho_m &= \mathrm{rewind}\left(s_{k_m} \to s_{j_m}\right), \qquad j_m < k_m, \\ \tau_{0:k_m} &= \underbrace{\tau_{0:j_m}}_{\text{retained prefix}} \oplus \underbrace{\beta_m}_{\text{abandoned suffix}}, \qquad \beta_m = \tau_{j_m:k_m}. \end{aligned} \tag{3}$$

Here, ρ_m denotes the rewind decision, before restoration is executed. The indices j_m and k_m refer to positions on the current execution path. The operator $\oplus$ denotes concatenation, with a shared boundary state included only once when joining trajectory segments.

Reflect. Before discarding β_m, the agent distills its useful information into a compact reflection:

$$e_m \sim \pi_\theta\left(\cdot \mid u, h_{j_m}, \beta_m, \mathcal{E}_m\right). \tag{4}$$

The reflection may capture discovered constraints, failure causes, actions to avoid, or alternative directions.

Restore and resume. Return to s_{j_m}, truncate history to h_{j_m}, and retain experience:

$$\mathcal{E}_{m+1} = \mathcal{E}_m \oplus e_m. \tag{5}$$

$$\left(s_{k_m}, h_{k_m}, \mathcal{E}_m\right) \longrightarrow \left(s_{j_m}, h_{j_m}, \mathcal{E}_{m+1}\right). \tag{6}$$

Execution resumes with the retained prefix and updated experience, allowing subsequent decisions to use a better-informed context from the same environment state. A complete trial of retrospective exploration is written as

$$\Gamma = \left(\sigma_0, \rho_0, e_0, \sigma_1, \rho_1, e_1, \ldots, \sigma_M\right), \tag{7}$$

where σ_m denotes a forward trajectory segment ending at a recovery boundary or trial termination. The first segment starts from s_0; each subsequent segment resumes from the selected earlier state and conditions on the updated experience. The abandoned suffix β_m is determined by the selected rewind point and may span portions of multiple earlier segments.

4 Learning from Retrospective Exploration

Retrospective exploration allows agents to revisit earlier states and continue with accumulated experience. Learning to use this capability effectively requires improving both execution decisions and the experience distilled from abandoned branches. It therefore yields two complementary learning signals. First, alternative continuations from the same environment state and retained prefix can be compared to identify more productive execution paths. Second, the progress of a resumed continuation can be compared with that of the abandoned suffix to assess the usefulness of the experience carried forward.

We use these two signals to jointly train a shared execution and reflection policy. **State-Anchored GRPO** implements the first comparison by assigning credit to continuations from the same state and

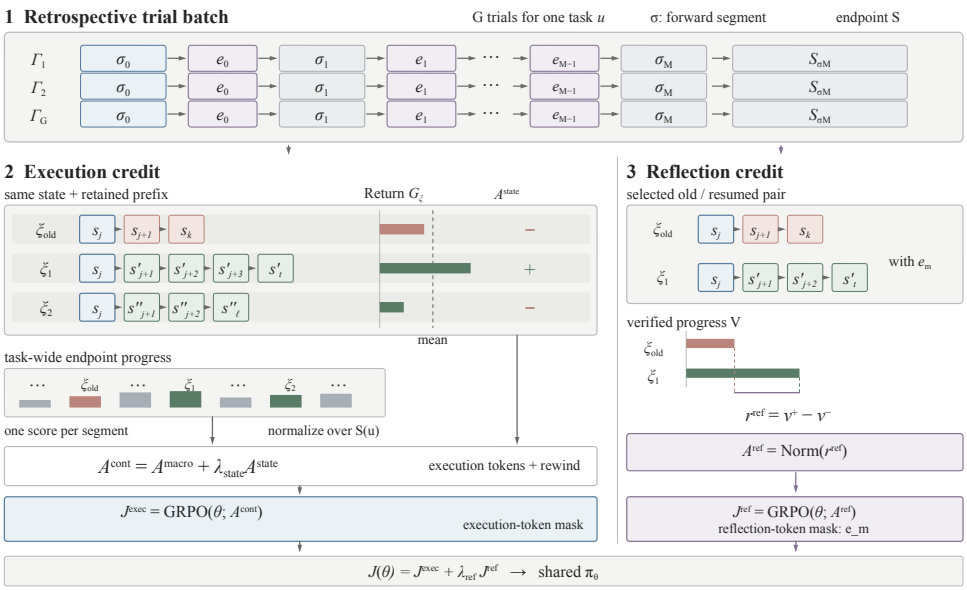

Figure 2: **Learning from retrospective exploration.** The top panel collects retrospective trials for one task. Left: State-Anchored GRPO combines task-wide endpoint progress with return comparisons among continuations sharing an environment state and retained prefix, producing A^{cont} for execution tokens including rewind. Right: the verified-progress difference between a matched abandoned/resumed pair produces A^{ref} for the reflection tokens that generate e_m. The separate masked objectives jointly update one shared policy.

retained prefix. A **branch-advancement reward** implements the second by measuring the progress of the resumed continuation relative to the abandoned suffix. Figure 2 illustrates these two learning signals.

4.1 State-Anchored GRPO over Revisited States

A final task outcome gives limited guidance about which decisions were productive. Local comparisons, as used in GiGPO (Feng et al., 2025a), provide more focused feedback. Retrospective exploration creates the comparison groups needed for execution credit: after returning to an earlier point, the agent can generate alternative continuations from the same environment state while retaining the same execution prefix.

Constructing continuation groups. For each task u, we sample a group of retrospective trials $\{\Gamma_i\}_{i=1}^{B}$ and collect their forward segments in $\mathcal{S}(u)$. Each recorded state visit defines a continuation ξ ending with its segment at success, failure, rewind, or budget exhaustion. A model-initiated rewind is included as the terminal execution decision when present.

Let s_ξ and h_ξ denote the starting environment state and retained history of ξ. We group continuations by the same task, starting state, and retained history:

$$\mathcal{H}_u(s,h) = \{\xi \in \mathcal{C}(u) : s_\xi = s,\ h_\xi = h\}\,, \tag{8}$$

where $\mathcal{C}(u)$ is the collection of continuations extracted from the sampled trials. Groups include earlier segment suffixes passing through s and new continuations after rewind. Samples retain their original generation history and experience; the accumulated experience may differ within a group.

Assigning execution credit. Each continuation has a discounted return G_ξ from its starting state to the segment endpoint and a task-progress score S_ξ at that endpoint, shared by continuations from the same segment. Its advantage combines task-wide and state-relative feedback:

$$A_\xi^{\text{cont}} = \underbrace{\text{Norm}_{\mathcal{S}(u)}(S_\xi)}_{A_\xi^{\text{macro}}} + \lambda_{\text{state}} \underbrace{\text{Norm}_{\mathcal{H}_u(s_\xi,h_\xi)}(G_\xi)}_{A_\xi^{\text{state}}}, \tag{9}$$

where Norm subtracts the corresponding group mean and divides by its standard deviation plus ϵ. The macro term normalizes endpoint progress over $\mathcal{S}(u)$, counting each segment once, to preserve the task-wide objective. The state-relative term favors higher-return continuations from the same state and prefix. For singleton groups, this term is zero and the macro signal remains.

We apply A^{cont}_{ξ} to the execution tokens in ξ, including any terminal model-initiated rewind decision. The retained prefix serves as input context and is excluded from the continuation loss. Execution returns end at rewind; subsequent advancement supervises the reflection response. Appendix D.5 details the construction, and Appendix D.6 illustrates this credit boundary.

4.2 Learning Reflection from Branch Advancement

Reflection is generated from an abandoned branch, but its value is realized only through the continuation that follows recovery. The discarded suffix may contain useful information about invalid actions, hidden constraints, or promising alternatives; however, the final trial outcome is too coarse and too delayed to determine whether this information improved the resumed exploration. A successful trial may succeed despite an uninformative reflection, while a useful reflection may produce measurable progress before task completion. We therefore evaluate reflection by comparing the progress of the post-recovery continuation with that of the abandoned branch from the same restored state.

At recovery m, the agent distills the abandoned suffix β_m into experience e_m and resumes from state s_{j_m}. The abandoned suffix and the new forward segment σ_{m+1} share this starting state, while the latter conditions on the updated experience $\mathcal{E}_{m+1}$. Here, β_m is the suffix discarded by the rewind and may contain portions of multiple earlier segments; it need not coincide with a single continuation used for execution grouping.

Using an environment-verifiable progress measure V, we evaluate the two paths:

$$v_m^- = V(\beta_m \mid s_{j_m}), \qquad v_m^+ = V(\sigma_{m+1} \mid s_{j_m}), \tag{10}$$

where V measures task progress, such as completed subgoals, for exploration starting from s_{j_m}. The branch-advancement reward uses their difference as a proxy for experience utility; positive values indicate greater progress after recovery:

$$r_m^{\mathrm{ref}} = v_m^+ - v_m^-. \tag{11}$$

Joint policy optimization. We jointly optimize execution and reflection through

$$\mathcal{J}(\theta) = \mathcal{J}^{\mathrm{exec}}_{\mathrm{GRPO}}\left(\theta; A^{\mathrm{cont}}\right) + \lambda_{\mathrm{ref}} \mathcal{J}^{\mathrm{ref}}_{\mathrm{GRPO}}\left(\theta; A^{\mathrm{ref}}\right), \tag{12}$$

where A^{ref} is obtained by group-normalizing the branch-advancement rewards r_m^{ref}, and λ_{ref} controls the relative weight of reflection training. The two objectives update π_θ through execution tokens (including model-initiated rewind) and reflection tokens, respectively. Appendix D.5 gives sample weighting and GRPO details.

5 Experiments

Our experiments ask whether ReTRAIL improves recovery-enabled success while retaining competitive forward execution (Section 5.2), which rollout and learning components support these gains (Section 6), and how performance varies with instance difficulty, task length, and the number of recovery attempts (Sections 5.3 and B.1).

5.1 Experimental Setup

Environments. The reported results cover four interactive benchmarks with complementary failure structures. **MineSweeper** (Li et al., 2024) requires logical deduction under partial observability; unsuccessful exploration may reveal board constraints. **Sokoban** (Racanière et al., 2017) tests planning with irreversible forward transitions, such as pushing a box into a dead corner. **ALFWorld** (Shridhar et al., 2021) evaluates multi-step instruction following, while **ScienceWorld** (Wang et al., 2022) involves procedural reasoning with long dependency chains. All use text interaction and the state-restoration interface in Section 3. Dataset partitions are listed in Table 5.

Table 1: Performance comparison on four long-horizon interactive benchmarks. **P@1** evaluates forward execution without rewind, **T@1** enables proactive rewind within one trial, and **T@3** permits up to three macro attempts with reflection-guided recovery. Every setting shares a 60-call budget per trial, including execution and reflection. Colored annotations show absolute changes relative to P@1; bold values indicate the best result in each column. All values are success rates (%).

Method	MineSweeper			Sokoban			ALFWorld			ScienceWorld		
	P@1	T@1	T@3	P@1	T@1	T@3	P@1	T@1	T@3	P@1	T@1	T@3
Prompting Methods												
Zero-shot	3.5	3.1$_{\downarrow 0.4}$	5.2$_{\uparrow 1.7}$	6.0	5.5$_{\downarrow 0.5}$	9.0$_{\uparrow 3.0}$	35.0	34.2$_{\downarrow 0.8}$	40.1$_{\uparrow 5.1}$	8.0	7.2$_{\downarrow 0.8}$	11.0$_{\uparrow 3.0}$
ReAct	6.0	6.5$_{\uparrow 0.5}$	10.2$_{\uparrow 4.2}$	7.0	6.8$_{\downarrow 0.2}$	11.8$_{\uparrow 4.8}$	40.0	41.0$_{\uparrow 1.0}$	47.5$_{\uparrow 7.5}$	10.2	9.4$_{\downarrow 0.8}$	15.1$_{\uparrow 4.9}$
Reflexion	5.5	7.0$_{\uparrow 1.5}$	9.8$_{\uparrow 4.3}$	6.5	6.2$_{\downarrow 0.3}$	12.0$_{\uparrow 5.5}$	38.0	42.1$_{\uparrow 4.1}$	45.3$_{\uparrow 7.3}$	9.0	13.0$_{\uparrow 4.0}$	16.2$_{\uparrow 7.2}$
Rollback	6.7	6.5$_{\downarrow 0.2}$	13.2$_{\uparrow 6.5}$	9.7	10.3$_{\uparrow 0.6}$	15.0$_{\uparrow 5.3}$	35.0	34.2$_{\downarrow 0.8}$	43.1$_{\uparrow 8.1}$	13.2	12.3$_{\downarrow 0.9}$	20.0$_{\uparrow 6.8}$
Fine-Tuning Methods												
GRPO	54.6	51.6$_{\downarrow 3.0}$	60.8$_{\uparrow 6.2}$	70.3	68.9$_{\downarrow 1.4}$	82.8$_{\uparrow 12.5}$	91.3	90.8$_{\downarrow 0.5}$	93.5$_{\uparrow 2.2}$	48.9	47.5$_{\downarrow 1.4}$	52.1$_{\uparrow 3.2}$
RLOO	53.2	50.4$_{\downarrow 2.8}$	59.1$_{\uparrow 5.9}$	69.1	67.8$_{\downarrow 1.3}$	81.0$_{\uparrow 11.9}$	90.7	90.2$_{\downarrow 0.5}$	92.8$_{\uparrow 2.1}$	47.6	46.3$_{\downarrow 1.3}$	50.8$_{\uparrow 3.2}$
GiGPO	57.9	56.4$_{\downarrow 1.5}$	65.5$_{\uparrow 7.6}$	**78.9**	79.7$_{\uparrow 0.8}$	85.3$_{\uparrow 6.4}$	90.9	90.5$_{\downarrow 0.4}$	94.3$_{\uparrow 3.4}$	56.1	55.0$_{\downarrow 1.1}$	58.3$_{\uparrow 2.2}$
BEACON	58.5	57.2$_{\downarrow 1.3}$	68.0$_{\uparrow 9.5}$	77.0	76.1$_{\downarrow 0.9}$	84.0$_{\uparrow 7.0}$	91.8	91.2$_{\downarrow 0.6}$	94.5$_{\uparrow 2.7}$	56.5	55.1$_{\downarrow 1.4}$	61.0$_{\uparrow 4.5}$
LaMer	55.4	57.2$_{\uparrow 1.8}$	70.8$_{\uparrow 15.4}$	72.1	70.5$_{\downarrow 1.6}$	83.3$_{\uparrow 11.2}$	89.7	90.1$_{\uparrow 0.4}$	95.7$_{\uparrow 6.0}$	53.5	54.2$_{\uparrow 0.7}$	60.2$_{\uparrow 6.7}$
ReTRAIL	**58.8**	**63.5**$_{\uparrow 4.7}$	**95.3**$_{\uparrow 36.5}$	76.6	**87.5**$_{\uparrow 10.9}$	**93.7**$_{\uparrow 17.1}$	**92.7**	**94.0**$_{\uparrow 1.3}$	**98.5**$_{\uparrow 5.8}$	**56.7**	**62.0**$_{\uparrow 5.3}$	**71.2**$_{\uparrow 14.5}$

Baselines. We compare with zero-shot prompting, ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2023), adapted GA-Rollback (Li et al., 2025), and trained baselines GRPO (Shao et al., 2024), RLOO (Ahmadian et al., 2024), GiGPO (Feng et al., 2025a), BEACON (Wang et al., 2026), and LaMer (Jiang et al., 2026). GiGPO uses local group comparisons; LaMer learns across episodes. Recovery-enabled evaluations share a rewind interface with corresponding execution and reflection prompts.

Evaluation protocol and metrics. **Pass@1 (P@1)** measures single-pass execution without rewind. **Trial@1 (T@1)** permits model-initiated rewind but terminates at the first irreversible failure; one macro-attempt may therefore explore multiple branches. $\Delta_{\text{pro}} = \text{T@1} - \text{P@1}$ measures the net gain from enabling proactive recovery. **Trial@3 (T@3)** permits up to three macro-attempts, with reflection-guided recovery after irreversible failure. We denote $\Delta_{\text{ref}} = \text{T@3} - \text{T@1}$ as the *retry gain*: it measures the combined benefit of additional attempts, recovery, and retained experience. The configured step limit for all settings is 60. Details are provided in Appendix C.4.

Training details. We use Qwen3-4B (Yang et al., 2025) as the base model for all methods. ReTRAIL samples groups of trials using retrospective exploration and is optimized using our State-Anchored GRPO objective (Section 4.1). For the standard RL baselines, we adjust the rollout group size to match the sampling budget of ReTRAIL at each gradient update. We otherwise keep the learning rate, batch size, prompt template, and task splits identical across methods. The continuation construction and execution objective are detailed in Appendix D.5. Appendix C.5 reports equal-benchmark aggregation. Reported results are means over three evaluation seeds.

5.2 Main Results

ReTRAIL achieves stronger performance. ReTRAIL achieves the highest T@1 and T@3 on all four benchmarks (Table 1), outperforming the strongest baseline in T@3 by 24.5 percentage points on MineSweeper and 10.2 on ScienceWorld. Its forward-only P@1 remains competitive, slightly exceeding the strongest baseline on MineSweeper, ALFWorld, and ScienceWorld, while trailing GiGPO by 2.3 points on Sokoban. Together, these results show that ReTRAIL's advantage becomes more pronounced when recovery is enabled, while its forward-solving ability remains competitive.

ReTRAIL makes effective use of test-time recovery. ReTRAIL benefits from recovery both within and across macro-attempts. Within a single attempt, proactive recovery yields positive gains on every benchmark, ranging from 1.3 percentage points on ALFWorld to 10.9 on Sokoban, whereas

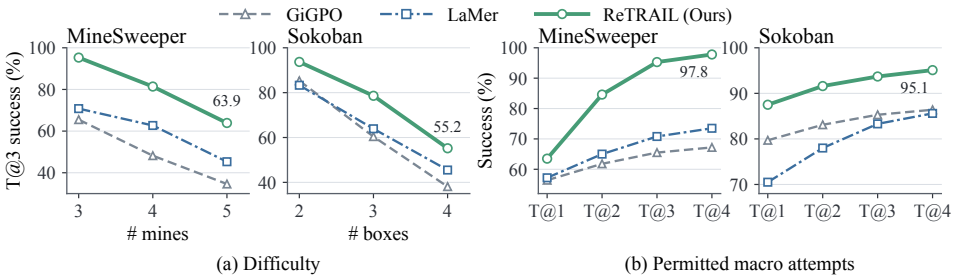

Figure 4: Scaling instance difficulty and recovery attempts. (a) T@3 success with additional mines or boxes; these settings vary instance difficulty. (b) Success with one to four permitted macro-attempts. ReTRAIL's endpoint value is annotated in each panel. Y-axis ranges differ by subplot and are explicitly labeled. Higher is better.

enabling rewind for baseline policies produces mixed changes. Across attempts, ReTRAIL further improves with reflection-guided recovery and retained experience. On MineSweeper, success rises from 63.5 at T@1 to 95.3 at T@3, a gain of 31.8 points, compared with 9.1 for GiGPO and 13.6 for LaMer. These results demonstrate ReTRAIL's ability to adapt through recovery, benefiting from both within-attempt course correction and continued exploration after failure.

ReTRAIL improves exploration coverage and execution efficiency. Figure 3 compares the coverage of distinct environment states and the number of actions taken in successful trials. Relative to LaMer, ReTRAIL increases the median number of distinct states visited from 11.0 to 14.0 on MineSweeper and from 8.0 to 11.5 on Sokoban. Meanwhile, the median action count among successful trials decreases from 53.0 to 40.0 and from 48.5 to 37.0, respectively. Together, these findings suggest that recovering early from unproductive branches helps ReTRAIL explore more broadly while completing tasks more efficiently.

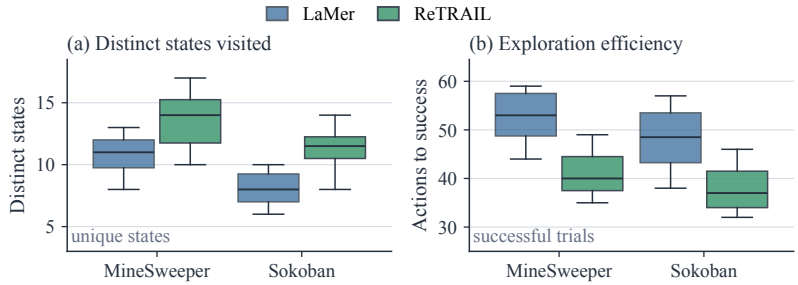

Figure 3: Exploration diversity and efficiency of LaMer and ReTRAIL on MineSweeper and Sokoban. (a) States visited, where higher values indicate broader exploration; (b) Actions to success, where lower values indicate more efficient solving. ReTRAIL visits more states while requiring fewer actions.

5.3 Scaling Instance Difficulty and Recovery Attempts

We next examine performance as instance difficulty and inference-time computation increase.

Scaling to harder instances. We increase the number of mines in MineSweeper and boxes in Sokoban to construct harder variants. Figure 4(a) shows that success decreases for all plotted methods, while ReTRAIL retains an advantage at every displayed setting. At the hardest setting, ReTRAIL reaches 63.9 on MineSweeper, compared with 45.3 for the strongest baseline, and 55.2 on Sokoban, compared with 45.5, yielding margins of 18.6 and 9.7 percentage points. These results show that ReTRAIL maintains its advantage on harder instances. The analysis by shortest-solution length is reported separately in Figure 5.

Table 2: Ablation of retrospective exploration. Success rate (%).

Variant	MineSweeper		Sokoban	
	T@1	T@3	T@1	T@3
ReTRAIL (full)	**63.5**	**95.3**	**87.5**	**93.7**
w/o model rewind	58.1	78.4	72.3	89.1
w/o reflection	61.8	69.7	86.0	87.2

Table 3: Ablation of the learning objective. Success rate (%).

Variant	MineSweeper		Sokoban	
	T@1	T@3	T@1	T@3
ReTRAIL (full)	**63.5**	**95.3**	**87.5**	**93.7**
w/o progress-diff reward	60.9	80.2	81.7	84.9
GiGPO objective	57.6	90.1	84.2	90.3

Scaling the number of attempts. Allowing up to four macro-attempts provides additional opportunities for recovery with accumulated experience. ReTRAIL achieves the highest success rate throughout Figure 4(b). On MineSweeper, success increases from 63.5 at T@1 to 97.8 at T@4, a gain of 34.3 percentage points, compared with 10.8 for GiGPO and 16.3 for LaMer. On Sokoban, ReTRAIL reaches 95.1 at T@4 and remains highest at each attempt budget. Thus, the learned execution and reflection policy continues to benefit from additional recovery opportunities.

6 Analysis

Ablation. Tables 2 and 3 evaluate the roles of retrospective exploration and its learning objectives. Removing model-initiated rewind lowers T@1 from 63.5 to 58.1 on MineSweeper and from 87.5 to 72.3 on Sokoban, showing the benefit of proactive recovery. Replacing distilled reflection with the raw abandoned suffix has a smaller effect on T@1, but lowers T@3 to 69.7 and 87.2, respectively. This suggests that distilling branch experience provides more useful guidance for subsequent attempts than retaining raw history alone.

We next ablate the learning objective. Replacing the branch-advancement reward $r_m^{\mathrm{ref}} = v_m^+ - v_m^-$ with post-recovery progress v_m^+ alone lowers T@3 from 95.3 to 80.2 on MineSweeper and from 93.7 to 84.9 on Sokoban, alongside smaller declines in T@1. These results support using subsequent branch progress to guide reflection. We also replace State-Anchored GRPO with GiGPO within the same retrospective-exploration framework; this variant is distinct from the standalone GiGPO baseline in Table 1. T@1/T@3 falls from $63.5/95.3$ to $57.6/90.1$ on MineSweeper and from $87.5/93.7$ to $84.2/90.3$ on Sokoban. Together, these results support the proposed execution and reflection objectives, with the GiGPO comparison evaluating the credit-assignment designs as implemented.

Recovery Gains across Task Horizons. We stratify tasks by reference-solution length to examine how ReTRAIL's recovery gains vary with task horizon (Appendix B.1). From short to long tasks, its reflection-guided retry gain increases from 4.5 to 19.0 percentage points, while its proactive gain rises from 1.3 to 7.8 points. Although LaMer achieves a slightly larger retry gain on short tasks, ReTRAIL surpasses it by 5.8 points on long tasks. These results suggest that retrospective exploration is particularly beneficial for long-horizon execution.

7 Conclusion

Learning to recover from unproductive exploration is crucial for autonomous agents to make sustained progress on long-horizon tasks. We introduced **ReTRAIL**, a framework that combines execution recovery with experiential learning through retrospective exploration. With an environment-level *rewind* interface, agents can revisit earlier states and explore alternative continuations while retaining experience from abandoned branches. ReTRAIL trains a shared execution and reflection policy under this protocol, using state-anchored comparisons to improve execution and branch-advancement feedback to guide reflection. Across four interactive benchmarks, ReTRAIL consistently outperforms both methods without rewind and baselines equipped with the same interface, with ablations supporting the complementary roles of recovery, retained experience, and progress-based feedback. More broadly, these findings highlight retrospective exploration as a promising foundation for autonomous agents that can recover from setbacks, adapt through accumulated experience, and sustain progress toward long-horizon goals.

AI USE STATEMENT

In preparing this manuscript, we used generative AI tools for language polishing and literature search and survey. Language assistance focused on improving grammar, sentence flow, clarity, and readability while preserving the intended technical meaning. Literature assistance supported the identification of relevant prior work and the organization and summarization of existing research. The authors remain responsible for reviewing all AI-assisted material, checking literature summaries and citations against the original sources, and ensuring that the manuscript accurately represents the research. We take full responsibility for the final content of this work, including all text, claims, and artifacts produced with the assistance of generative AI.

ETHICS STATEMENT

This work studies recovery and learning in autonomous agents through experiments on four controlled benchmark environments: MineSweeper, Sokoban, ALFWorld, and ScienceWorld. The framework aims to improve agents' ability to recover from unproductive exploration, but state restoration in these environments should not be interpreted as reversing the consequences of actions in real-world systems. Deployment beyond these benchmarks should therefore include appropriate authorization, safeguards for external actions, and human oversight where actions may have consequential or irreversible effects. Retained interaction histories and reflection memories should be handled with appropriate privacy and access controls, and any reuse or release of data, models, and software should respect their applicable licenses and terms of use.

REPRODUCIBILITY STATEMENT

To support reproducibility, we specify the retrospective-exploration framework and learning objectives in Sections 3 and 4. Appendix D details environment restoration, recovery and context management, and provides pseudocode for rollout collection and policy updates. Section 5.1 and Appendix C document the benchmarks and data partitions, baseline configurations, training hyperparameters, reward and progress signals, evaluation protocols, model-call accounting, and aggregation across benchmarks and seeds. Appendix E provides execution and reflection prompt templates and examples of observation formatting for the four benchmark environments.

REFERENCES

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms, 2024. URL https://arxiv.org/abs/2402.14740.

Anthropic. Introducing claude opus 4.5. https://www.anthropic.com/news/claude-opus-4-5, November 2025. Accessed: 2026-04-29.

Chaithanya Bandi, Ben Hertzberg, Geobio Boo, Tejas Polakam, Jeff Da, Sami Hassaan, Manasi Sharma, Andrew Park, Ernesto Hernandez, Dan Rambado, Ivan Salazar, Rafael Cruz, Chetan Rane, Ben Levin, Brad Kenstler, and Bing Liu. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers, 2026. URL https://arxiv.org/abs/2602.00933.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Adithya Bhaskar, Xi Ye, and Danqi Chen. Language models that think, chat better. *CoRR*, abs/2509.20357, 2025.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin,

Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

Arthur Chen, Zuxin Liu, Jianguo Zhang, Akshara Prabhakar, Zhiwei Liu, Shelby Heinecke, Silvio Savarese, Victor Zhong, and Caiming Xiong. Test-time adaptation for llm agents via environment interaction, 2026a. URL https://arxiv.org/abs/2511.04847.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Generating instruction manuals by LLM agents via interactive environmental learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=Pwl9n4zlf5.

Xiaoyin Chen, Canwen Xu, Yite Wang, Boyi Liu, Zhewei Yao, and Yuxiong He. Learning to self-evolve. *arXiv preprint arXiv:2603.18620*, 2026b. URL https://arxiv.org/abs/2603.18620.

Yuxin Chen, Yu Wang, Yi Zhang, Ziang Ye, Zhengzhou Cai, Yaorui Shi, Qi Gu, Hui Su, Xunliang Cai, Xiang Wang, An Zhang, and Tat-Seng Chua. Learning to self-verify makes language models better reasoners, 2026c. URL https://arxiv.org/abs/2602.07594.

Jae-Woo Choi, Hyungmin Kim, Hyobin Ong, Youngwoo Yoon, Minsu Jang, Dohyung Kim, and Jaehong Kim. ReAcTree: Hierarchical LLM agent trees with control flow for long-horizon task planning. In *International Conference on Autonomous Agents and Multiagent Systems*, 2026. URL https://arxiv.org/abs/2511.02424.

Victor de Lamo Castrillo, Habtom Kahsay Gidey, Alexander Lenz, and Alois Knoll. Fundamentals of building autonomous llm agents, 2025. URL https://arxiv.org/abs/2510.09244.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948, 2025.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Erhang Li, Fangqi Zhou, Fangyun Lin, Fucong Dai, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoran Wei, Haowei Zhang, Haowen Luo, Haozhe Ji, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, Jialiang Huang, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jingchang Chen, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexin Huang, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Liang Zhao, Liangsheng Yin, Lihua Guo, Lingxiao Luo, Linwang Ma, Litong Wang, Liyue Zhang, M. S. Di, M. Y Xu, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Panpan Huang, Peixin Cong, Peiyi Wang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, S. H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Songyang Zhou, Tao Ni, Tao Yun, Tian Pei, Tian Ye, Tianyuan Yue, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjun Gao, Wentao Zhang, Xi Gao, Xiangwen Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyuan Li, Xu Chen, Xuecheng Su, Xuehai Pan, Xuheng Lin, Xuwei Fu, Y. Q. Wang, Yang Zhang, Yanhong Xu, Yanru Ma, Yao Li, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yiliang Xiong, Ying He, Ying Zhou, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuduan Wang, Yue Gong, Yuhan Wu, Yuheng Zou, Yukun Li, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehua Zhao, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhiyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Zizheng Pan, Zongqing Yao, Bei Feng, Hui Li, J. L. Cai, Jiaqi Ni,

Lei Xu, Meng Li, Ning Tian, R. J. Chen, R. L. Jin, S. S. Li, Shuang Zhou, Tianyu Sun, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xinnan Song, Xinyi Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, Dongjie Ji, Jian Liang, Jianzhong Guo, Jin Chen, Leyi Xia, Miaojun Wang, Mingming Li, Peng Zhang, Ruyi Chen, Shangmian Sun, Shaoqing Wu, Shengfeng Ye, T. Wang, W. L. Xiao, Wei An, Xianzu Wang, Xiaowen Sun, Xiaoxiang Wang, Ying Tang, Yukun Zha, Zekai Zhang, Zhe Ju, Zhen Zhang, and Zihua Qu. Deepseek-v3.2: Pushing the frontier of open large language models, 2025.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/420c9f777c0b4f78d515e53cf74d58b2-Abstract-Conference.html.

Sicheng Feng, Gongfan Fang, Xinyin Ma, and Xinchao Wang. Efficient reasoning models: A survey. *Trans. Mach. Learn. Res.*, 2025, 2025b.

Google. A new era of intelligence with gemini 3, 2025. URL https://blog.google/products-and-platforms/products/gemini/gemini-3/#note-from-ceo.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *CoRR*, abs/2507.17746, 2025.

Izzeddin Gur, Hiroki Furuta, Austin V Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=9JQtrumvg8.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *ACL (1)*, pp. 3828–3850. Association for Computational Linguistics, 2024.

Shuo He, Lang Feng, Qi Wei, Xin Cheng, Lei Feng, and Bo An. Hierarchy-of-groups policy optimization for long-horizon agentic tasks. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=T8Dev99qnz.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi GU, Hui Su, and Xunliang Cai. Vitabench: Benchmarking LLM agents with versatile interactive tasks in real-world applications. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=rtcX9qOBaz.

Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. EvoTest: Evolutionary test-time learning for self-improving agentic systems. In *International Conference on Learning Representations*, 2026c. URL https://arxiv.org/abs/2510.13220.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Wese: Weak exploration to strong exploitation for llm agents, 2024. URL https://arxiv.org/abs/2404.07456.

Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, et al. Verltool: Towards holistic agentic reinforcement learning with tool use. *arXiv preprint arXiv:2509.01055*, 2025.

Yulun Jiang, Liangze Jiang, Damien Teney, Michael Moor, and Maria Brbić. Meta-RL induces exploration in language agents. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2512.16848.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *CoRR*, abs/2503.09516, 2025.

Xingzuo Li, Kehai Chen, Yunfei Long, Xuefeng Bai, Yong Xu, and Min Zhang. Generator-assistant stepwise rollback framework for large language model agent, 2025. URL https://arxiv.org/abs/2503.02519.

Yinghao Li, Haorui Wang, and Chao Zhang. Assessing logical puzzle solving in large language models: Insights from a minesweeper case study. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 59–81. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.naacl-long.4. URL http://dx.doi.org/10.18653/v1/2024.naacl-long.4.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*. OpenReview.net, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Yuxin Liu, Ziang Ye, Yueqing Sun, Mingye Zhu, Jinwei Xiao, Zhuowen Han, Qi GU, Xunliang Cai, and Lei Zhang. Map: A map-then-act paradigm for long-horizon interactive agent reasoning, 2026. URL https://arxiv.org/abs/2605.13037.

Zhanzhi Lou, Hui Chen, Yibo Li, Qian Wang, and Bryan Hooi. Learning to learn-at-test-time: Language agents with learnable adaptation policies. *arXiv preprint arXiv:2604.00830*, 2026. URL https://arxiv.org/abs/2604.00830.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning, 2025. URL https://arxiv.org/abs/2308.08747.

Shinji Mai, Yunpeng Zhai, Ziqian Chen, Cheng Chen, Anni Zou, Shuchang Tao, Zhaoyang Liu, and Bolin Ding. Cues: A curiosity-driven and environment-grounded synthesis framework for agentic rl, December 2025. URL https://arxiv.org/abs/2512.01311.

MiniMax. Minimax m2 and agent: Ingenious in simplicity, 2025. URL https://www.minimax.io/news/minimax-m2.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

OpenAI. Introducing gpt-5.2, 2025. URL https://openai.com/index/introducing-gpt-5-2/.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.25140.

Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Hassan Awadallah. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6300–6323, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.326. URL https://aclanthology.org/2025.findings-acl.326/.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=tBRNC6YemY.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Sébastien Racanière, Théophane Weber, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

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

Zhihong Shao, Yuxiang Luo, Chengda Lu, Z. Z. Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *CoRR*, abs/2511.22570, 2025.

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 *EuroSys*, pp. 1279–1297. ACM, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

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

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis, 2026. URL https://arxiv.org/abs/2601.05808.

Hongjin SU, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan O Arik. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=3UKOzGWCVY.

Yi Su, Dian Yu, Linfeng Song, Juntao Li, Haitao Mi, Zhaopeng Tu, Min Zhang, and Dong Yu. Crossing the reward bridge: Expanding RL with verifiable rewards across diverse domains. *CoRR*, abs/2503.23829, 2025.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. Stop overthinking: A survey on efficient reasoning for large language models. *Trans. Mach. Learn. Res.*, 2025, 2025.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon agent via context folding. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=lNRgWoGfYg.

Yiyou Sun, Xinyang Han, Weichen Zhang, Yuanbo Pang, Tianyu Wang, Yuhan Cao, Yixiao Huang, Chris Duroiu, Haoyun Zhang, Jeffrey Lin, Weishu Zhang, Tyler Zeng, Ying Yan, Bo Liu, Hanson Wen, Mingyang Xu, Xiaoyuan Liu, Zimeng Chen, Weiyan Shi, Amanda Dsouza, Vincent Sunn Chen, Patrick Bryant, Carl Boettiger, Yamini Rangan, Bradley Rothenberg, Kyle Steinfeld, Arvind Rao, Tapio Schneider, Georgios Yannakakis, Laure Zanna, Kaan Ozbay, Ida Sim, Tarek Zohdi, George Em Karniadakis, Jack Gallant, Teresa Head-Gordon, Yushan Li, Wenxi Deng, Tao Sun, Huiqi Wang, Zhun Wang, Justin Xu, Chris Yuhao Liu, Yafei Cheng, Rongwang Hu, Aras Bacho, Shengcao Cao, Zengyi Qin, Yixiong Chen, Hengduan Fan, Hao Liu, Lin Zeng, Shashank Muralidhar Bharadwaj, Litian Gong, Yingxuan Yang, Maojia Song, Ruheng Wang, Zongzheng Zhang, Honglin Bao, Shuo Lu, Jianhong Tu, Zhonghua Wang, Zheng Zhang, Zijiao Chen, Yanqiong Jiang, Zhendong Li, Bohan Lyu, Chang Ma, Peiran Xu, Benran Zhang, Shangding Gu, Haoyue Hua, Haoyang Li, Wanzhe Liao, Chengzhi Liu, Junbo Peng, Haoran Sun, Zechen Xu, Bo Chen, Jiayi Cheng, Yi Jiang, Keying Kuang, Yuan Li, Youbang Pan, Ziyan Rao, Alexander Schubert, Yifan Shen, Vincent Siu, Xiatao Sun, Kangqi Zhang, Xiaopan Zhang, Yuchen Zhu, Ishaan Singh Chandok, Lei Ding, Jingxuan Fan, Andrew Glover, Jiaming Hu, Yiran Hu, Wenbo Huang, Zixin Jiang, Haoran Jin, Lukas Kim, Ming Liu, Yang Liu, Alireza Rafiei, Xuhuan Shen, Kunyang Sun, Sophia Sun, Ting Sun, Eric Wang, Yixin Wang, Hanwen Xing, Sihan Xu, Yuzheng Xu, Zhongxing Xu, Zhiling Yan, Boqin Yuan, Ruiqi Zhang, Yifan Zhang, Zibo Zhao, Liana, Santanu Bosu Antu, Haoyue Bai, Carlo Bosio, Joseph Cavanagh, Patricia Cavazos-Rehg, Tianxing Chen, Xuewen Chen, Yipu Chen, Chenyu Zhu, Chen Dai, Stefano De Castro, Yunfu Deng, Kaustubh Dhole, Jiayuan Ding, Chenchen Du, Zhehang Du, Hao Fan, Run-Ze Fan, Hengyu Fu, Shi Gu, Yifan Gu, Charlie Guo, Baihe Huang, Baixiang Huang, Rimika Jaiswal, Zhihan Jiang, Ran Jin, Erin Kasson, Xin Lan, Joseph Lee, Deren Lei, Chenyu Li, Daofeng Li, Haitao Li, Hongwei Li, Jingyan Li, Xiao Li, Yi Li, Yinsheng Li, Yuangang Li, Zhixu Li, Wenyu Liang, Longtai Liao, Kevin Qinghong Lin, Andy Zeyi Liu, Che Liu, Jiaming Liu, Kaiyuan Liu, Xuan Liu, Pan Lu, Wenbo Lv, Yicheng Lyu, Qiuyang Mang, Kyle Montgomery, Yuzhou Nie, Ruoxi Ning, Jorin Overwiening, Xu Pan, Layna Paraboschi, Core Francisco Park, Justin Purnomo, Swati Rajwal, Scott Rankin, Bixuan Ren, Yiren Rong, HaoYang Shang, Ventus Shaw, Fiona Shen, Jiawei Shen, Minqi Shi, Shi Qiu, Huaxiu Yao, Tianneng Shi, Jonah So, Vladislav Susoy, Hannah Szlyk, Haocheng Wang, Jialu Wang, Wei Wang, Xinyu Wang, Zehao Wang, Dowling Wong, Angela Wu, Dehao Wu, Fangyu Wu, Mengyuan "Millie" Wu, Yu Wu, Yuchen Wu, Yuhao Wu, Qingpo Wuwu, Weihang Xiao, Yongyi Xiong, Fan Xu, Ruiling Xu, Mingxuan Yan, Benjamin Yang, Jirong Yang, Sen Yang, Xiaoli Yang, Yushi Yang, Haoran Ye, Xiaohu Yu, Zhengming Yu, Chenlong Zhang, Chi Zhang, Hanning Zhang, Hanwen Zhang, Junge Zhang, Kunpeng Zhang, Song Zhang, Wenjin Zhang, Wenshuo Zhang, Ying Zhang, Yizhi Zhang, Brian Zhao, Qijian Zhao, Yimin Zhao, Yuhaohua Zheng, Liwei Zhou, Tianyue Zhou, Sichen Zhu, Siqi Zhu, Yan Zhu, Yishu Zhu, Jierui Zuo, Chonghao Cai, Helena Casademunt, Wenjia Chen, Cheng Cheng, Nawen Deng, Rao Fu, Tianfu Fu, Yifan Han, He Ren, Zhenyu He, Qiao Jin, Langlang Li, Yuetai Li, Sylvia Liu, Lu Lu, Luqing Zhou, Subhabrata Mukherjee, Yunqi Ouyang, Yin Ren, Dawei Shi, Haoran Wu, Zhiyue Wu, Hannah Yao, Zhuoran Yi, Jenny Yu, Rhea Zhan, Hang Zhou, Blake Zhu, Junfan Zhu, Alan Yuille, Yang Liu, Russell Alan Poldrack, Jiachen Li, Zhenglu Li, Molei Tao, Jing Huang, Wenqi Shi, Costas Spanos, Lichao Sun, Chenguang Wang, Orson Xu, Zhen Dong, Hector Gomez, Aylin Caliskan, Ali Emami, Haimin Hu, Zhi Li, Lihui Liu, Murphy Niu, Yi Shao, Jianxin Sun, Mikko Tolonen, Ting Wang, Sanjiv Das, Yanjun Gao, Wenbo Guo, Erika J Schneider, Zhiyong Lu, Yian Ma, Mark Mueller, Radha Poovendran, Somayeh Sojoudi, Yinglun Zhu, and Dawn Song. Agents' last exam, 2026b. URL https://arxiv.org/abs/2606.05405.

Gemma Team. Gemma 3 technical report. *CoRR*, abs/2503.19786, 2025.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pp. 5998–6008, 2017.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Rui Wang, Hongru Wang, Boyang Xue, Jianhui Pang, Shudong Liu, Yi Chen, Jiahao Qiu, Derek Fai Wong, Heng Ji, and Kam-Fai Wong. Harnessing the reasoning economy: A survey of efficient reasoning for large language models. *CoRR*, abs/2503.24377, 2025a.

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 (EMNLP)*, pp. 11279–11298. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-main.775. URL https://aclanthology.org/2022.emnlp-main.775/.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: understanding self-evolution in LLM agents via multi-turn reinforcement learning. *CoRR*, abs/2504.20073, 2025b.

Zixuan Wang, Yuchen Yan, Hongxing Li, Teng Pan, Dingming Li, Ruiqing Zhang, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Milestone-guided policy learning for long-horizon language agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2605.06078.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents, 2025. URL https://arxiv.org/abs/2504.12516.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. Os-atlas: A foundation action model for generalist gui agents, 2024. URL https://arxiv.org/abs/2410.23218.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. Agentgym: Evolving large language model-based agents across diverse environments, 2024.

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

Teng Xiao, Yige Yuan, Hamish Ivison, Huaisheng Zhu, Faeze Brahman, Nathan Lambert, Pradeep Dasigi, Noah A. Smith, and Hannaneh Hajishirzi. Meta-reinforcement learning with self-reflection for agentic search. *arXiv preprint arXiv:2603.11327*, 2026. URL https://arxiv.org/abs/2603.11327.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 2024. doi: 10.48550/ARXIV.2404.07972. URL https://doi.org/10.48550/arXiv.2404.07972.

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, 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. *CoRR*, abs/2412.15115, 2024.

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

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2207.01206.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Ziang Ye, Yang Zhang, Wentao Shi, Xiaoyu You, Fuli Feng, and Tat-Seng Chua. Visualtrap: A stealthy backdoor attack on gui agents via visual grounding manipulation, 2025a. URL https://arxiv.org/abs/2507.06899.

Ziang Ye, Zhenru Zhang, Yang Zhang, Jianxin Ma, Junyang Lin, and Fuli Feng. Disentangling reasoning tokens and boilerplate tokens for language model fine-tuning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20939–20957, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1078. URL https://aclanthology.org/2025.findings-acl.1078/.

Ziang Ye, Wentao Shi, Yuxin Liu, Yu Wang, Zhengzhou Cai, Yaorui Shi, Qi Gu, Xunliang Cai, and Fuli Feng. Look before you leap: Autonomous exploration for llm agents, 2026. URL https://arxiv.org/abs/2605.16143.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Weinan Dai, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: an open-source LLM reinforcement learning system at scale. *CoRR*, abs/2503.14476, 2025a.

Tianyu Yu, Bo Ji, Shouli Wang, Shu Yao, Zefan Wang, Ganqu Cui, Lifan Yuan, Ning Ding, Yuan Yao, Zhiyuan Liu, Maosong Sun, and Tat-Seng Chua. RLPR: extrapolating RLVR to general domains without verifiers. *CoRR*, abs/2506.18254, 2025b.

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

Z.AI. Glm-4.7: Advancing the coding capability, 2025. URL https://z.ai/blog/glm-4.7.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms, 2023.

Yuyuan Zeng, Yufei Huang, Can Xu, Qingfeng Sun, Jianfeng Yan, Guanghui Xu, Tao Yang, and Fengzong Lian. Zero reinforcement learning towards general domains. *CoRR*, abs/2510.25528, 2025.

Hanchen Zhang, Xiao Liu, Bowen Lv, Xueqiao Sun, Bohao Jing, Iat Long Iong, Zhenyu Hou, Zehan Qi, Hanyu Lai, Yifan Xu, Rui Lu, Hongning Wang, Jie Tang, and Yuxiao Dong. Agentrl: Scaling agentic reinforcement learning with a multi-turn, multi-task framework, 2025. URL https://arxiv.org/abs/2510.04206.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges, 2024. URL https://arxiv.org/abs/2401.07339.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

Zhisong Zhang, Tianqing Fang, Kaixin Ma, Wenhao Yu, Hongming Zhang, Haitao Mi, and Dong Yu. WebRollback: Enhancing web agents with explicit rollback mechanisms. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 187–197. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-short.12. URL https://aclanthology.org/2026.eacl-short.12/.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=5NTt8GFjUHkr.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 62138–62160. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *ArXiv*, abs/2307.13854, 2023. URL https://api.semanticscholar.org/CorpusID:260164780.

Siyu Zhou, Tianyi Zhou, Yijun Yang, Guodong Long, Deheng Ye, Jing Jiang, and Chengqi Zhang. WALL-e: World alignment by neurosymbolic learning improves world model-based LLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=DorAT49sxj.

Yifei Zhou, Sergey Levine, Jason Weston, Xian Li, and Sainbayar Sukhbaatar. Self-challenging language model agents, 2025. URL https://arxiv.org/abs/2506.01716.

A Limitations and Future Work

Our results motivate two directions for extending retrospective exploration.

(i) Extending recovery to broader environments. The current framework assumes access to state restoration through snapshots or reset-and-replay; the latter requires deterministic transitions and resettable external state (Appendix D.1). Extending recovery to environments with approximate restoration or irreversible side effects is an important next step. Future work could explore restoration controllers and compensating actions, together with checks on the recovered state, to determine when earlier progress can be safely reused.

(ii) Improving recovery credit and efficiency. ReTRAIL combines task-wide execution feedback with branch advancement, an environment-progress proxy for the utility of retained experience. The current execution return ends at each rewind boundary, while subsequent advancement supervises reflection (Appendix D.6). This design motivates studying credit assignment across recovery boundaries, so that rewind timing and target selection can receive direct feedback from the resumed branch. Accounting for restoration and reflection costs, and developing progress estimators for settings with sparse or noisy feedback, could further broaden the framework's practical use.

B Additional Results and Analysis

B.1 Recovery Gains by Reference-Solution Length

We group held-out instances by reference-solution length from the initial state. All methods use the same groups, independent of their observed rollout lengths.

Reference-solution length and grouping. For Sokoban, the reference is the breadth-first-search solution length d_0 (Appendix C.2); for MineSweeper, it is the minimum reveal count under correct deduction. For ALFWorld and ScienceWorld, it is the length of the provided expert or gold action sequence. We use the term *reference-solution length* because expert sequences need not be shortest solutions. On ALFWorld, we follow the length boundaries of Wang et al. (2026): Short (at most 4 actions), Medium (5–7), and Long (more than 7). For each remaining environment, we use the terciles of its held-out reference lengths. The bins therefore describe length within each environment, rather than a common absolute action count across environments.

Metrics and aggregation. Within each length bin, we compute the proactive gain $\Delta_{\mathrm{pro}} = \mathrm{T@1} - \mathrm{P@1}$ and the reflection-guided retry gain $\Delta_{\mathrm{ref}} = \mathrm{T@3} - \mathrm{T@1}$ on the same instances for every method. We then average each gain across the four environments with equal weight; values are success-rate differences in percentage points.

Observed pattern. ReTRAIL's retry gain increases from $+4.5$ on short tasks to $+9.2$ on medium and $+19.0$ on long tasks; LaMer's corresponding gains are $+5.6$, $+6.0$, and $+13.2$. Thus, ReTRAIL's retry-gain advantage over LaMer is -1.1, $+3.2$, and $+5.8$ points in the three groups. Its proactive gain also increases from $+1.3$ to $+5.3$ and $+7.8$. These descriptive results show larger recovery gains for ReTRAIL on longer-reference tasks, while LaMer has the larger retry gain in the short group. They do not isolate length from other task properties or establish a causal effect of reflection alone.

B.2 Average Performance Across Benchmarks

Table 4 summarizes Table 1 using the equal-benchmark rule in Appendix C.5. It is computed from the reported, rounded benchmark means, so minor rounding differences are possible. The reported spread is across evaluation seeds, not additional runs. The forward-only results support competitive performance, without establishing statistically significant superiority on each benchmark.

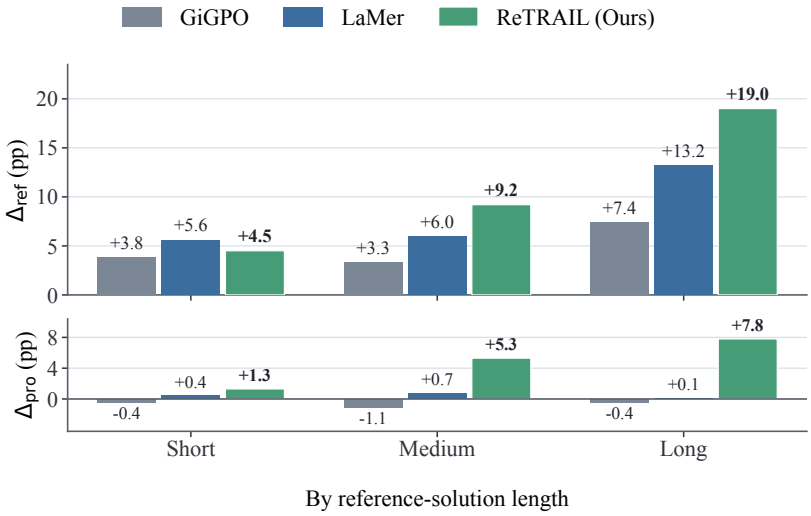

Figure 5: Recovery gains by reference-solution length. The upper row shows reflection-guided retry gain $\Delta_{\text{ref}} = \text{T@3} - \text{T@1}$; the lower row shows proactive gain $\Delta_{\text{pro}} = \text{T@1} - \text{P@1}$. Values are equal-weight averages across the four environments, in percentage points. Short, Medium, and Long use the environment-specific grouping rules above.

Table 4: Equal-weight averages across the four benchmarks, computed from the reported benchmark means in Table 1 and rounded to one decimal place. $\pm$ is the standard deviation across the three evaluation seeds. These are aggregate point estimates, not additional runs.

Method	P@1	T@1	T@3
Zero-shot	13.1 $\pm$1.6	12.5 $\pm$1.7	16.3 $\pm$2.7
ReAct	15.8 $\pm$2.9	15.9 $\pm$1.2	21.2 $\pm$3.2
Reflexion	14.8 $\pm$2.5	17.1 $\pm$2.6	20.8 $\pm$1.4
Rollback	16.2 $\pm$2.7	15.8 $\pm$1.7	22.8 $\pm$3.1
GRPO	66.3 $\pm$1.2	64.7 $\pm$1.4	72.3 $\pm$2.0
RLOO	65.2 $\pm$2.3	63.7 $\pm$2.3	70.9 $\pm$2.1
GiGPO	71.0 $\pm$1.7	70.4 $\pm$2.0	75.9 $\pm$1.9
BEACON	71.0 $\pm$2.5	69.9 $\pm$1.1	76.9 $\pm$2.0
LaMer	67.7 $\pm$0.9	68.0 $\pm$2.2	77.5 $\pm$2.4
ReTRAIL	**71.2** $\pm$1.5	**76.8** $\pm$1.9	**89.7** $\pm$2.2

C Experimental Setup

C.1 Benchmarks and Data Partitions

All four benchmarks use text interaction and an environment-restoration interface. Table 5 lists the registered task instances loaded by the training entrypoints. The registered Sokoban variants use 6×6 or 7×7 rooms with two or three boxes; each has 256 training and 128 test entries. The difficulty experiment varies the box count separately. Dataset counts refer to task entries, not repeated rollouts or evaluation seeds.

C.2 Rewards and Progress Signals

The optimizer uses three quantities: the per-decision reward δ_t, the segment-endpoint score S, and the branch score $V(\xi \mid s)$. Table 6 gives their definitions. If one decision executes several primitive actions, δ_t sums their environment rewards. A decision with no environment transition, including a rewind request, has $\delta_t = 0$.

Table 5: Registered task-instance partitions loaded by the training entrypoints. Counts describe distinct dataset entries, not generated continuations or repeated evaluations. ALFWorld's selected test partition is `valid_seen`; its separately registered `valid_unseen` partition contains 134 entries. ScienceWorld uses the L0 partition.

Environment	Train	Test	Partition information
MineSweeper	256	128	6×6 boards, 3 mines
Sokoban	256	128	registered room variants share these sizes
ALFWorld	3,553	140	`train` / `valid_seen`
ScienceWorld	3,322	1,661	L0 task/variation pairs

Table 6: Training signals by environment. δ_t supplies the execution return, S the task-wide endpoint score, and V the branch score for reflection. Derived advantages and branch-score differences are not policy inputs.

	δ_t (per step)	S (segment endpoint)	$V(\xi \mid s)$ (branch)
MineSweeper	native board reward: $+0.5$ safe reveal, $+2.0$ blank-region expansion, $+10.0$ board cleared, -0.1 mine revealed, -1.0 invalid or no-change action	cumulative δ on the retained path from s_0	cumulative δ along the branch
Sokoban	environment-returned transition reward	$d_0 - d(\text{endpoint})$ by breadth-first search	$d_0 - d_{\text{end}}$, with $d_{\text{end}} = 0$ if solved and $d_{\text{end}} = d_0$ if deadlocked
ALFWorld	environment-returned transition reward	fraction of satisfied goal conditions at the endpoint	maximum fraction of satisfied goal conditions along the branch
ScienceWorld	environment-returned transition reward	native completion score at the endpoint	maximum native completion score along the branch
Policy visibility	environment-dependent native feedback	ScienceWorld score is visible	the branch comparison is not a prompt input

MineSweeper. Completion and mine revelation end the attempt, with the corresponding terminal reward in Table 6 applied to that action. S accumulates rewards along the retained path from s_0; restoring a checkpoint also restores this prefix score. V sums rewards only after the selected anchor, so the common prefix cancels in the branch comparison.

Sokoban. Let d_0 be the shortest-solution distance from the initial state and d_{end} the remaining distance at a branch endpoint, both computed by breadth-first search. We use $V = d_0 - d_{\text{end}}$, taking $d_{\text{end}} = 0$ for a solved state and $d_{\text{end}} = d_0$ for a deadlocked or unsolvable state. The new-minus-old branch score is therefore $d^-_{\text{end}} - d^+_{\text{end}}$. These search distances are evaluator-only and are not shown to the policy.

ALFWorld and ScienceWorld. ALFWorld measures the fraction of satisfied goal conditions; ScienceWorld uses the native task-completion score. In each environment, S is the endpoint score and V is the maximum score along the branch, including its starting state. These differ when progress can be undone. ScienceWorld's native score is visible in its observations; the derived branch-advancement reward is not.

C.3 Training Configuration and Sampling Budget

All trained methods start from Qwen3-4B (Yang et al., 2025). They share the optimizer, learning-rate schedule, task splits, and decoding settings. Table 7 lists the configuration. Prompt serialization follows each rollout interface (Appendix E); shared task instructions do not imply identical execution and reflection conversations.

Table 7: Training and evaluation configuration of ReTRAIL. Optimizer settings are shared with the trained baselines. The retrospective-exploration and objective blocks describe ReTRAIL; a trial can contain several forward segments and should not be equated with one baseline trajectory.

Setting	**Value**
Policy and optimization	
Learning rate	1×10^{-6}
LR Scheduler	Constant
Optimizer updates	300
Batch size	16
Advantage-normalization ϵ	1×10^{-6}
Discount γ	1.0
Retrospective exploration rollout	
Baseline trajectories per task/update	24
ReTRAIL trials per task/update B	8
Macro-attempts per trial (training)	at most 3
Model calls per ReTRAIL trial, execution + reflection	60
Model calls per baseline trajectory	60
Maximum calls per task/update, baseline / ReTRAIL	1440 / 480
Model-initiated rewinds per trial	at most 3
Context length	16,384
Objective weights	
State-anchored weight λ_{state}	0.5
Reflection weight λ_{ref}	0.5
Evaluation	
Evaluation seeds	3, reported as the mean

Table 8: Units used for rollout and budget accounting.

Unit	Meaning
Trial	The full run under one global model-call budget.
Macro-attempt	One attempt until success, irreversible failure, or budget exhaustion; proactive rewinds stay within it.
Forward segment	Execution between recovery boundaries, or until success, failure, or budget exhaustion.
Model call	One execution or reflection generation; a rewind request is an execution call.
Primitive action	One environment action; a Sokoban execution call can contain several moves.

Per task and update, baselines sample 24 trajectories, each allowing 60 model calls. ReTRAIL samples eight trials, each allowing 60 calls and at most three macro-attempts. This aligns nominal attempt slots, with call ceilings of 1,440 for the baseline and 480 for ReTRAIL. These ceilings do not measure realized calls, tokens, or compute. Experience and advantage-normalization groups remain within each task.

C.4 Evaluation Protocol and Call Accounting

Table 8 distinguishes the units used throughout the paper. Rewinding restores an active path but never refunds calls already made.

P@1 disables recovery. T@1 allows proactive recovery within one macro-attempt and stops at the first irreversible failure. T@3 also permits failure-triggered recovery, up to three macro-attempts. Training uses the same attempt cap. Model-initiated rewinds are limited to three per trial. The attempt-scaling experiment varies the macro-attempt cap from one to four.

Training and evaluation impose a global ceiling of 60 model calls per trial: $p + f \leq 60$, where p counts execution and f reflection. A proactive recovery uses an execution call to request rewind and a separate reflection call. Forced recovery uses a reflection call after the preceding execution. Recovery starts only when its attempt/rewind allowance permits it and at least two calls remain for

reflection and continuation. Reset-and-replay without model generation adds environment actions but no model calls.

The prompt backends also track a branch allowance. MineSweeper and Sokoban restore its saved consumption at the checkpoint; the supplied ALFWorld flow resets its segment-turn counter. These backend rules are distinct from the global call ceiling. A forward-segment cap counts both recovery types, whereas the macro-attempt count increases only after failure-triggered recovery. Algorithm 1 takes these rules as configuration inputs.

C.5 Aggregation Across Benchmarks and Seeds

Let $p_{s,e}^{(k)}$ be success on benchmark e, under evaluation seed s and protocol $k \in \{\mathrm{P@1}, \mathrm{T@1}, \mathrm{T@3}\}$. We average equally across the four benchmarks and three seeds:

$$M_s^{(k)} = \frac{1}{4}\sum_{e=1}^{4} p_{s,e}^{(k)}, \qquad \overline{M}^{(k)} = \frac{1}{3}\sum_{s=1}^{3} M_s^{(k)} = \frac{1}{4}\sum_{e=1}^{4} \overline{p}_e^{(k)}. \tag{13}$$

Each benchmark has equal weight regardless of its instance count. This differs from pooling all test instances. Table 4 reports the resulting means together with the standard deviation across the three evaluation seeds. Individual seed values cannot be recovered from these means.

D Method Details

This section follows the data flow: restore an environment state, resume execution with branch memory, record the rollout, and construct the training objective.

D.1 Environment Restoration and Checkpoints

The environment harness implements rewind. MineSweeper and Sokoban store and restore serializable state snapshots. ALFWorld and ScienceWorld reset the task and replay the recorded actions up to the selected point, since their engines do not expose a general snapshot interface. Replay requires deterministic transitions and resettable external state.

The active path stores ordered checkpoints $\mathcal{K} = (C_0, \ldots, C_k)$, each containing its state, path history, and restoration information. A target $j < k$ identifies a path position: identical observations can occur at different positions. Restoring C_j truncates the active path but preserves the chronological training archive Γ. This archive stores all outputs with their original input contexts, segment boundaries, feedback, and abandoned/resumed branch pairs. Thus an abandoned branch remains available for training after leaving the live prompt.

D.2 Recovery Triggers and Reflection

A model-initiated recovery begins with an execution output requesting rewind. A forced recovery begins when the controller detects a configured failure. Both generate reflection before restoration, while the abandoned branch is still available. The reflection returns branch memory in `<remark>` tags.

For MineSweeper and Sokoban, a valid model-selected target uses a known-target reflection prompt; forced recovery asks the model to select a checkpoint as well. ALFWorld and ScienceWorld also request a checkpoint in their reflection responses. Thus *forced* identifies who triggers recovery, not a fixed destination immediately before failure. Table 9 summarizes the two paths; Appendix C.4 specifies their budget limits.

D.3 Context After Recovery

The logical history h is the retained execution path. The actual input c is the conversation produced by the environment's message builder. The distinction matters because the backends serialize history differently.

Table 9: **Recovery triggers and model calls.** Restoration follows reflection, so the abandoned branch remains available when its memory is generated. Checkpoint selection and fallback parsing follow the environment-specific prompt implementation.

	Model-initiated recovery	Forced recovery
Trigger	execution output requests an earlier checkpoint	controller observes a configured failure condition
Target	requested by the policy; validated before restoration	selected through the checkpoint-choice reflection interface
Reflection	a separate generation returning `<remark>`	a separate generation returning `<remark>` and a target
Call accounting	one execution call for the request, plus one reflection call	one reflection call in addition to preceding execution calls
Main-text metric	available under T@1 and T@3	failure-triggered retries available under T@3

MineSweeper and Sokoban save a conversation at each checkpoint. Recovery restores this prefix, removes the abandoned continuation, and appends a user message with the new branch memory. A previous trailing reflection at the same checkpoint is first made available to the reflection call, then replaced. The next observation is appended as another user message.

ScienceWorld also restores a message prefix, but includes branch memories in its subsequent observation. ALFWorld rebuilds the resumed conversation from the system message, restored observation, admissible commands, and branch memories; it does not copy the earlier conversation prefix. Appendix E gives the literal templates and formatting examples. Training always uses the recorded input c, without adding later reflections to earlier samples.

D.4 Rollout Algorithm

Algorithm 1 records one trial under a frozen behavior policy. Its four blocks follow Section 3. The active history h, checkpoints $\mathcal{K}$, and memory $\mathcal{E}$ guide execution; the archive Γ retains every generation with its original context and feedback. A segment σ groups execution decisions between recovery boundaries, and $n = p + f$ counts all model calls.

Invalid outputs include unparseable actions and disabled or out-of-range rewind requests; they consume a call and receive feedback without changing the environment. Checkpoints include states reached on failure. Recovery requires allowance for reflection and continuation (Appendix C.4). Target parsing follows Appendix D.2; memory and context updates follow Appendix D.3. Restoring a checkpoint never truncates Γ or refunds calls.

D.5 State-Anchored GRPO

Each execution decision is one training sample. Its remaining segment supplies the return used to score that decision, while the loss applies only to its own generated output.

D.5.1 Decision Samples and Returns

For task u, collect B trials $\{\Gamma_i\}_{i=1}^{B}$ and their forward segments $\mathcal{S}(u)$. Index decision t in segment $\sigma_{i,m}$ by $q = (i, m, t)$, and let $T_{i,m}$ be the segment length:

$$\mathcal{I}(u) = \{(i, m, t) : \sigma_{i,m} \in \mathcal{S}(u),\ 0 \le t < T_{i,m}\}, \qquad \xi_q = \sigma_{i,m}[t : T_{i,m}]. \tag{14}$$

The continuation ξ_q ends with its original segment, at success, failure, rewind, or budget exhaustion. A proactive rewind is the segment's final execution decision. Record its starting state s_q, logical retained history h_q, and actual input context. Let $\mathcal{Y}_q$ contain only the generated reasoning and action or rewind request for decision q. The indexed collection $\mathcal{C}(u) = (\xi_q)_{q \in \mathcal{I}(u)}$ contains return records, not extra sequences to train again.

For example, an old segment $s_0 \to s_1 \to s_2 \to s_3$ that ends in a rewind to s_1 supplies a continuation from s_1, even though the segment began at s_0. This can be compared with the new continuation from

Algorithm 1 Retrospective exploration for one task

Require: task u, frozen policy π_{old}, protocol $\mathcal{B}$, call limit $C_{\max} = 60$
reset task; initialize $h, \mathcal{K} = (C_0), \mathcal{E} = \varnothing$ and protocol counters
$\Gamma \leftarrow \varnothing$; $\sigma \leftarrow \varnothing$; $n \leftarrow 0$
while $n < C_{\max}$ and $\mathcal{B}$ permits execution **do**
    1. Execute Equation 2
    $c \leftarrow \textsc{BuildContext}(u, h, \mathcal{E}, \mathcal{K})$; $y \sim \pi_{\text{old}}(\cdot \mid c)$; $n \leftarrow n + 1$
    record (c, y) in Γ; append the execution decision to σ
    if y is invalid **then** record feedback; **continue**
    2. Detect recovery Equation 3
    if y requests rewind **then**
        retain the requested target ▷ model-initiated recovery
    else
        execute the parsed action; update $h, \mathcal{K}$; record rewards and observations in Γ
        if task solved **then break**
        if no failure trigger **then continue** ▷ otherwise, forced recovery
    end if
    close σ in Γ
    if $n + 2 > C_{\max}$ or $\mathcal{B}$ disallows recovery/continuation **then break**
    3. Reflect Equation 4
    build c_{ref} from the active branch, memory, and any requested target
    $z \sim \pi_{\text{old}}(\cdot \mid c_{\text{ref}})$; $n \leftarrow n + 1$; record (c_{ref}, z) in Γ
    resolve (j, e) using any requested target, z, and the configured fallback
    if no valid target $0 \le j < |\mathcal{K}| - 1$ remains **then break**
    4. Restore and resume Equation 6
    record abandoned suffix $\beta = \tau_{j:k}$ and recovery link in Γ, where $k = |\mathcal{K}| - 1$
    restore C_j; truncate $h, \mathcal{K}$ to the retained prefix
    update $\mathcal{E}$ and resumed context with e; update branch/protocol counters
    start a new empty σ linked to this recovery
end while
close any nonempty open segment in Γ; **return** Γ

s_1. The two decisions share a state and logical prefix, but retain their original generation contexts and potentially different branch memories.

D.5.2 Execution Advantages

The return from decision q and its comparison group are

$$G_q = \sum_{\ell=t}^{T_{i,m}-1} \gamma^{\ell-t} \delta_{i,m,\ell}, \qquad \mathcal{H}_u(s,h) = \{q \in \mathcal{I}(u) : s_q = s,\ h_q = h\}, \tag{15}$$

Here $\delta_{i,m,\ell}$ sums environment rewards produced by execution decision ℓ. A rewind request has zero transition reward; reflection calls do not enter this return. Group members share a task, environment state, and retained history. Their accumulated experience and remaining global budget may differ, so the comparison does not hold the full policy input fixed.

The macro signal uses endpoint progress $S_{i,m}$, with each segment contributing once to normalization regardless of its length. For any decision in that segment, the main text's S_{ξ_q} equals $S_{i,m}$. The combined advantage is

$$\begin{aligned} A_q^{\text{state}} &= \text{Norm}_{\mathcal{H}_u(s_q,h_q)}(G_q), \\ A_q^{\text{cont}} &= \underbrace{\text{Norm}_{\mathcal{S}(u)}(S_{i,m})}_{A_{i,m}^{\text{macro}}} + \lambda_{\text{state}} A_q^{\text{state}}. \end{aligned} \tag{16}$$

Norm subtracts the group mean and divides by its standard deviation plus ϵ. Singleton and constant-score groups have zero advantage; empty groups contribute no samples. These rules apply separately to macro, state, and reflection groups within each task. When a decision has no local comparison, its macro term remains available.

All decisions in a segment share its macro advantage but have their own local advantages. Each advantage applies only to that decision's output. A reward can contribute to several earlier returns without duplicating output tokens in the loss. Repeated visits are separate generation events.

D.5.3 Reflection Rewards and Eligibility

At a recovery point, reflection is scored by the improvement of the resumed branch over the abandoned suffix:

$$r_m^{\mathrm{ref}} = V(\sigma_{m+1} \mid s_{j_m}) - V(\beta_m \mid s_{j_m}). \tag{17}$$

The evaluator computes V using the environment-specific definitions in Appendix C.2. Let $\mathcal{Y}_{i,m}^{\mathrm{ref}}$ be the reflection output. Eligible recoveries are

$$\mathcal{R}(u) = \big\{(i,m) :\ \mathcal{Y}_{i,m}^{\mathrm{ref}} \neq \varnothing,\ T_{i,m+1} > 0,\ v_{i,m}^{-}, v_{i,m}^{+} \text{ are defined}\big\}. \tag{18}$$

Here $v_{i,m}^{-}$ and $v_{i,m}^{+}$ are the abandoned and resumed branch scores. A missing resumed segment has length zero. Eligibility requires an observed target, not positive advancement or eventual success. Both recovery types are normalized together over $\mathcal{R}(u)$ within the same task. If no continuation is recorded, the reflection tokens receive no loss; earlier execution outputs keep their usual contribution.

D.5.4 Loss Masks and Policy Update

Execution and reflection use disjoint output masks. A rewind request made during execution belongs to $\mathcal{Y}_q$; a checkpoint choice generated within a reflection belongs to that reflection. Prompts, observations, copied memories, and retained-prefix tokens are masked out.

Let $\mathcal{Y}_{\mathrm{ref}}(u)$ contain the eligible reflection tokens, with advantages A^{ref} normalized over $\mathcal{R}(u)$. The task objective is

$$\begin{aligned}\mathcal{J}_u(\theta) = \frac{1}{\max\{1, |\mathcal{I}(u)|\}} \sum_{q \in \mathcal{I}(u)} \mathcal{J}_{\mathrm{GRPO}}\big(\theta; \mathcal{Y}_q, A_q^{\mathrm{cont}}\big) \\ + \lambda_{\mathrm{ref}} \mathcal{J}_{\mathrm{GRPO}}\big(\theta; \mathcal{Y}_{\mathrm{ref}}(u), A^{\mathrm{ref}}\big)\,,\end{aligned} \tag{19}$$

$\mathcal{J}_{\mathrm{GRPO}}$ is the token-averaged clipped policy surrogate. The execution term first averages tokens within a decision, then averages over decisions. An empty reflection set or a term with no contributing tokens gives zero loss. Both current and behavior policies use each token's original context. The batch combines task objectives without pooling their normalization groups.

Algorithm 2 summarizes the update. All rollouts are collected before optimization; the behavior policy, data, and advantages remain fixed throughout the corresponding update.

Algorithm 2 One State-Anchored GRPO update with task-local groups

Require: nonempty task batch $\mathcal{U}$, current parameters θ, B trials per task, weights $\lambda_{\mathrm{state}}, \lambda_{\mathrm{ref}}$
$\theta_{\mathrm{old}} \leftarrow \theta$ ▷ freeze the behavior policy for this batch
for each task $u \in \mathcal{U}$ **do**
  sample $\{\Gamma_i\}_{i=1}^{B}$ with $\pi_{\theta_{\mathrm{old}}}$ using Algorithm 1
  collect segments $\mathcal{S}(u)$, decisions $\mathcal{I}(u)$, and return records $\mathcal{C}(u)$
  retain each decision's context, behavior log-probabilities, and output mask
  compute $S_{i,m}$ once per segment and return-to-go G_q for each decision
  normalize $\{S_{i,m}\}_{\sigma_{i,m} \in \mathcal{S}(u)}$ to obtain $A_{i,m}^{\mathrm{macro}}$
  group visits by (s_q, h_q) within u; compute A_q^{state} and A_q^{cont} by Equation 16
  form the eligible recovery set $\mathcal{R}(u)$ in Equation 18
  for each $(i,m) \in \mathcal{R}(u)$ **do**
    $r_{i,m}^{\mathrm{ref}} \leftarrow V(\sigma_{i,m+1} \mid s_{j_{i,m}}) - V(\beta_{i,m} \mid s_{j_{i,m}})$
  end for
  normalize $\{r_{i,m}^{\mathrm{ref}}\}_{(i,m) \in \mathcal{R}(u)}$ to obtain $A_{i,m}^{\mathrm{ref}}$
  form $\mathcal{J}_u(\theta)$ with each execution output once (Equation 19)
end for
update θ from task objectives; keep θ_{old}, data, and advantages fixed

D.6 Credit Assignment at a Rewind: An Analytic Example

Consider action a with reward d, followed by proactive rewind ρ. Their decision indices are q_a, q_ρ, and their shared macro advantage is $M = \mathrm{Norm}_{\mathcal{S}(u)}(S_{i,m})$. Rewind ends the segment with zero transition reward:

$$\begin{aligned} G_{q_a} &= d + \gamma\, 0 = d, & G_{q_\rho} &= 0, \\ A^{\mathrm{cont}}_{q_a} &= M + \lambda_{\mathrm{state}}\, \mathrm{Norm}_{\mathcal{H}_u(s_{q_a}, h_{q_a})}(d), & A^{\mathrm{cont}}_{q_\rho} &= M + \lambda_{\mathrm{state}}\, \mathrm{Norm}_{\mathcal{H}_u(s_{q_\rho}, h_{q_\rho})}(0). \end{aligned} \tag{20}$$

Each advantage trains its own output. A singleton rewind group gives total advantage M. With group returns $\{0, g\}$, $g > 0$, its local contribution is $-\lambda_{\mathrm{state}}(g/2)/(\mathrm{Std}\{0, g\} + \epsilon)$. The total remains positive if M is large enough. Comparison with negative returns can instead give a positive local term.

With abandoned score b and resumed score $b + \Delta$, reflection receives

$$r^{\mathrm{ref}}_{i,m} = (b + \Delta) - b = \Delta, \qquad A^{\mathrm{ref}}_{i,m} = \mathrm{Norm}_{\mathcal{R}(u)}(\Delta). \tag{21}$$

This trains the whole reflection, without separating target and memory credit. Normalization can give positive advancement a negative advantage if other reflections improve more; a singleton group gives zero.

Execution returns stop at rewind. With the pre-rewind segment and group statistics fixed, different proactive targets receive the same execution advantage regardless of later outcomes. Post-recovery feedback applies to the separate reflection response.

E Prompt Templates

This section groups the execution, reflection, and observation templates by environment.

Reflection is a separate model call. Appendix D.3 explains how each backend assembles the resumed context; Appendix C.4 defines the global call ceiling. Literal references to "model action steps" below do not allow extra calls.

The observation boxes are formatting examples, not measured trajectories. They use checkpoint C_1, one action used, a global action allowance of 60, and a branch allowance of 20. Sokoban allows three moves per turn; ScienceWorld uses score zero. Invalid-action feedback, prior memories, and recent-event blocks are omitted from these examples and are added by the builders when present.

E.1 MineSweeper

The `refacted`, `undiscounted_final`, and `gigpo` flows share the following system and reflection templates.

MineSweeper Rewind System Prompt

```
You are an expert agent operating in the Minesweeper game with REWIND capability.
You will be given a {board_size} by {board_size} board, with {n_mines} hidden mines.
Rows and columns are indexed from 1 to {board_size}.

The task is guaranteed to be solvable. You must never conclude that the task is unsolvable,
↪  hopeless, or impossible, and you must never give up. Even if some branches appear
↪ unpromising or lead to dead ends, you can still learn from them, rewind when necessary,
↪ and continue exploring.
Your environment state may move backward through rewind, but your knowledge must keep
↪ moving forward. Use each failed attempt to refine your understanding, avoid repeating
↪ the same mistakes, and make better decisions in future branches until the task is solved
↪ .


# Cell States
- Unopened cells (?): hidden cells that may contain mines.
- Blank cells (.): opened safe cells with no neighboring mines.
- Numbered cells (1-8): opened safe cells showing how many neighboring cells contain mines.
- Mine cells (*): opened mine cells; revealing one loses the game.

# Goal
Reveal all Unopened and non-mine cells without hitting any mine.
```

```
# Reveal Rules
- Choose exactly ONE unopened cell (?) to reveal per model action step.
- Blank cells auto-cascade to reveal contiguous blanks and bordering numbered cells, you
↪ did not need to reveal the blank cells.
- Numbered cells reveal only that single cell.
- Mine cells end the current branch in failure.

# Checkpoints and Rewind Capability
You operate under two budgets, and you should manage them differently.


The first is a global action budget: {step_budget} model action steps for the entire
↪ episode. Each model action step reveals exactly one cell. This is a hard ceiling shared
↪ across the whole episode. You may use it for exploration, trial and error, and
↪ ultimately solving the task. This budget is accumulated globally and is never refunded
↪ by rewinding.

The second is a task-solve budget: {branch_attempt_budget} model action steps for making
↪ progress along the current active path. You should aim to solve the task within this
↪ allowance. Unlike the global action budget, this budget is tied to the checkpoint path.
↪ When you rewind to checkpoint C_j, the task-solve budget is rolled back to the amount
↪ already consumed at C_j. In other words, rewinding to C_j refunds the task-solve steps
↪ spent after C_j, while everything consumed up to C_j remains spent.

Use the global action budget and the task-solve budget together to balance exploration,
↪ learning from failed branches, trial and error, and direct progress toward solving the
↪ task.

Checkpoints track the current path as C_0, C_1, ..., C_k:
- C_0 is the initial board state.
- C_i is the state reached after i executed reveal actions on the current path.

Once you have executed at least one reveal action, you may travel back to any
earlier checkpoint:
- `<action>rewind to C_j</action>`, where 0 <= j < the current checkpoint index.
- `<action>rewind to j</action>` is equivalent.

Rewinding restores the environment to checkpoint C_j and rolls your branch attempt budget
back to its value at C_j, but what you learned on the abandoned branch is not lost | it
persists as memory. you can rewind when you find
* a better plan to start at C_j
* recognize the current branch is stuck(an action trap, an invalid loop, a dead end, or
↪ unproductive exploration)
* You are in good path to success but find insufficient budget to make progress(the task-
↪ solve budget is exhausted) and want to move to current state in less turns and context.

# Response Format
- First reason step-by-step about numbered clues, candidate mines, and safe cells.
- Then reveal one unopened (?) cell(number cell and Blank cells is safe but make no
↪ progress): `<action>(row, col)</action>`.
- OR rewind to a previous checkpoint: `<action>rewind to C_j</action>`.
Only the final `<action>...</action>` tag will be executed.
```

MineSweeper Known-Target Reflection Prompt

```
You are an expert Minesweeper player reflecting on a branch attempt.
Board: {board_size} by {board_size}, {n_mines} mines. Rows/columns indexed 1 to {board_size
↪ }.

# Cell States
- Unopened cells (?): hidden cells that may contain mines.
- Blank cells (.): opened safe cells with no neighboring mines.
- Numbered cells (1-8): opened safe cells showing how many neighboring cells contain mines.
- Mine cells (*): opened mine cells; revealing one loses the game.

# Reveal Rules
- Choose exactly ONE unopened cell (?) to reveal per model action step.
- Blank cells auto-cascade to reveal contiguous blanks and bordering numbered cells, you
↪ did not need to reveal the blank cells.
- Numbered cells reveal only that single cell.
- Mine cells end the current branch in failure.

# Core rollback semantics
The game was at step C_{rewind_from_step} when you decided to rewind back to step C_{
↪ rewind_to_step}.
```

```
Thus the env state returns to step C_{rewind_to_step}, Your goal is to push the knowledge
↪ state forward.
compress the branch attempt into a small branch memory that record what have you done that
↪ lead to what result,
what have you learned, what you should avoid and what you should do next.
Do not preserve the full trajectory. Preserve only information useful for future decisions
the core target is to help move closer to the target during the next new attempt or try
↪ explore other states.

# Context
{rewind_target_instruction}

# Rewind trigger
{rewind_reason}

# Relevant checkpoint states
{checkpoint_context}

# State at C_{rewind_from_step} before rewinding
{current_observation}

# Branch history from C_{rewind_to_step} to C_{rewind_from_step}
{history_after_rewind}

# Your task
Reflect on what went wrong and produce a concise branch memory for the next attempt based
↪ on the actions you have done and the environment feedback.
Focus on:
- revealed mine cells that should not be repeated;
- invalid/already-open/out-of-range reveals;
- numbered clue constraints and specific cells that are likely mines or safe;
- whether the branch should restart before or after a useful clue reveal;
- a concrete improved plan from the checkpoint you return to. you could move faster as you
↪ already know  some of the state and the knowledges.

Include a compact branch memory inside <remark> </remark> tags.
Use this structure, replacing C_j with the checkpoint you choose or confirm:
<remark>
# History Attempt from C_{rewind_to_step} to C_{rewind_from_step}
State at C_{rewind_to_step}:
{rewind_to_observation}
State at C_{rewind_from_step}:
{rewind_from_observation}

Actions I have done: ...
What's bad about the actions: ...
How to move forward: ...
Next new plan from C_j that can move faster and avoid the same mistakes: ...
</remark>
```

MineSweeper Observation-Message Example (user role)

```
Current observation at checkpoint C_1 (segment turn 2):
{observation}

Global action budget used: 1/60 (remaining: 59).
Task-solve budget used on this branch: 1/20 (remaining: 19).
Current checkpoint: C_1.
Valid model rewind targets: C_0.
Choose exactly one unopened cell to reveal inside one final action tag, e.g. <action>(2, 3)
↪ </action>, or travel back: <action>rewind to C_j</action>.
```

E.2 SOKOBAN

The `refacted`, `undiscounted_final`, and `gigpo` flows share the following system and reflection templates.

Sokoban Rewind System Prompt

```
You are an expert Sokoban agent operating with REWIND capability.
The task is guaranteed to be solvable. You must never conclude that the task is unsolvable,
↪  hopeless, or impossible, and you must never give up. Even if some branches appear
↪ unpromising or lead to dead ends, you can still learn from them, rewind when necessary,
↪ and continue exploring.
```

```
Your environment state may move backward through rewind, but your knowledge must keep
↪ moving forward. Use each failed attempt to refine your understanding, avoid repeating
↪ the same mistakes, and make better decisions in future branches until the task is solved
↪ .

# Symbols and Their Meaning
- Walls (`#`): These block movement.
- Floor (`_`): Open spaces where you can walk and move boxes.
- Targets (`O`): The spots where boxes need to go.
- Boxes (`X`): These are what you need to push onto the targets.
- Player (`P`): That's you!
- Box on Target (`√`): A box successfully placed on a target.
- Player on Target (`S`): You standing on a target.

# Goal
Push all boxes (`X`) onto targets (`O`).

# Rules
- Admissible actions: ["up", "down", "left", "right"]
- You can only push one box at a time. You can't pull boxes.
- You can't walk through or push boxes into walls or other boxes.
- Avoid pushing boxes into corners or against walls where they can't be moved.
- A blocked move may consume budget without improving the state, so avoid repeating it.

# Checkpoints and Rewind Capability
You operate under two budgets, and you should manage them differently.

The first is a global action budget: {step_budget} model action steps for the entire
↪ episode. Each model action step may contain up to {num_actions_per_turn} primitive
↪ Sokoban moves. This is a hard ceiling shared across the whole episode. You may use it
↪ for exploration, trial and error, and ultimately solving the task. This budget is
↪ accumulated globally and is never refunded by rewinding.

The second is a task-solve budget: {branch_attempt_budget} model action steps for making
↪ progress along the current active path. You should aim to solve the task within this
↪ allowance. Unlike the global action budget, this budget is tied to the checkpoint path.
↪ When you rewind to checkpoint C_j, the task-solve budget is rolled back to the amount
↪ already consumed at C_j. In other words, rewinding to C_j refunds the task-solve steps
↪ spent after C_j, while everything consumed up to C_j remains spent.

Use the global action budget and the task-solve budget together to balance exploration,
↪ learning from failed branches, trial and error, and direct progress toward solving the
↪ task.

Checkpoints track your progress along the current path as C_0, C_1, ..., C_k:
- C_0 is the initial state.
- C_i is the state reached after i executed model action steps on this path.

Once you have executed at least one model action step, you may travel back to any earlier
checkpoint:
- `<action>rewind to C_j</action>`, where 0 <= j < the current checkpoint index.
- `<action>rewind to j</action>` is equivalent.

Rewinding restores the environment to checkpoint C_j and rolls your branch attempt budget
back to its value at C_j, but what you learned on the abandoned branch is not lost | it
persists as memory. you can rewind when you find
* a better plan to start at C_j
* recognize the current branch is stuck(an action trap, an invalid loop, a dead end, or
↪ unproductive exploration)
* You are in good path to success but find insufficient budget to make progress(the task-
↪ solve budget is exhausted) and want to move to current state in less turns and context(
↪ more actions each turn)

# Response Format
- First reason step-by-step about the current state and box/target geometry.
- Then choose up to {num_actions_per_turn} moves inside one final action tag, e.g.
  `<action>up, right, right</action>`.
- OR rewind to a previous checkpoint: `<action>rewind to C_j</action>`.
Only the final `<action>...</action>` tag will be executed.
```

Sokoban Known-Target Reflection Prompt

```
You are an expert Sokoban player independently reflecting on a branch attempt.

# Symbols and Their Meaning
- Walls (`#`): These block movement.
- Floor (`_`): Open spaces where you can walk and move boxes.
- Targets (`O`): The spots where boxes need to go.
```

```
- Boxes (`X`): These are what you need to push onto the targets.
- Player (`P`): The agent.
- Box on Target (`√`): A box successfully placed on a target.
- Player on Target (`S`): Agent standing on a target.

# Rules
- Admissible actions: ["up", "down", "left", "right"]
- You can only push one box at a time. You can't pull boxes.
- You can't walk through or push boxes into walls or other boxes.
- Avoid pushing boxes into corners or against walls where they can't be moved.
- A blocked move may consume budget without improving the state, so avoid repeating it.


# Core rollback semantics
The game was at step C_{rewind_from_step} when you decided to rewind back to step C_{
↪ rewind_to_step}.

Thus the env state returns to step C_{rewind_to_step}, Your goal is to push the knowledge
↪ state forward.
compress the branch attempt into a small branch memory that record what have you done that
↪ lead to what result,
what have you learned, what you should avoid and what you should do next.
Do not preserve the full trajectory. Preserve only information useful for future decisions
the core target is to help move closer to the target during the next new attempt or try
↪ explore other states.

# Context
{rewind_target_context}

# Rewind trigger
{rewind_reason}

# Relevant checkpoint states
{checkpoint_context}

# State at C_{rewind_from_step} before rewinding
{current_observation}

# Branch history from C_{rewind_to_step} to C_{rewind_from_step}
{history_after_rewind}

# Prior reflection outputs
{reflection_history_context}

# Your task
Reflect on the branch attempt and produce a concise branch memory for the next new attempt
↪ direction from the checkpoint you return to.


Focus on:
- Invaild actions that environment not accepted or skipped by the environment or stuck by
↪ the walls or boxes;
- failure reason or likely bad push/walk sequence;
- boxes pushed into corners, walls, dead zones, or away from reachable targets;
- blocked/no-op moves or invalid loops that should not be repeated;
- useful spatial facts about player access, box positions, target positions, and required
↪ order;
- the task is always solvable, so you should always find a concrete improved plan from the
↪ checkpoint you return to.

Include a compact branch memory inside <remark> </remark> tags.
Use this structure, replacing C_j with the selected checkpoint:
<remark>
# History Attempt from C_{rewind_to_step} to C_{rewind_from_step}
State at C_{rewind_to_step}:
{rewind_target_context}
State at C_{rewind_from_step}:
{current_observation}
Actions I have done: ...
What's bad about the actions: ...
How to move forward: ...
Next new Plan from C_j that Must be different from the actions I have done
and independently and carefully think about and forward towards the goal and to a different
↪  state or to a good observed state but in less turns and context(more actions each turn)
↪ : ...
</remark>
```

Sokoban Observation-Message Example (user role)

```
Current observation at checkpoint C_1 (segment turn 2):
{observation}

Global action budget used: 1/60 (remaining: 59).
Task-solve budget used on this branch: 1/20 (remaining: 19).
Current checkpoint: C_1.
Valid model rewind targets: C_0.
Choose up to THREE moves inside one final action tag, e.g. <action>up, right</action>, or
↪ travel back: <action>rewind to C_j</action>.
```

E.3 ALFWORLD

Reward variants wrap the same rewind-choice flow. The system prompt below extends the household prompt with explicit rewind instructions. The user-message builder adds checkpoints, budgets, admissible commands, and branch memories. Actions and rewind requests use an `action` code block.

ALFWorld Rewind System Prompt

```
You are an expert agent controlling a robot in the ALFWorld household environment with
↪ REWIND capability. Your task is to complete household tasks by navigating rooms and
↪ interacting with objects.

The task is guaranteed to be solvable. You must never conclude that the task is unsolvable,
↪  hopeless, or impossible, and you must never give up. Even if some branches appear
↪ unpromising or lead to dead ends, you can still learn from them, rewind when necessary,
↪ and continue exploring.
Your environment state may move backward through rewind, but your knowledge must keep
↪ moving forward. Use each failed attempt to refine your understanding, avoid repeating
↪ the same mistakes, and make better decisions in future branches until the task is solved
↪ .

Action Commands:
For an environment action, choose one of the following, strictly following the command (
↪ argument) format. To rewind instead, use the checkpoint command described below.

## Navigation & Observation:
- look: Look around your current location to get more details.
- inventory: Check the object you are currently holding (you can only hold one).
- go to (receptacle): Move to a receptacle (e.g., table, fridge, sink).

## Interacting with Receptacles:
- open (receptacle): Open a receptacle.
- close (receptacle): Close a receptacle.

## Interacting with Objects:
- take (object) from (receptacle): Pick up an object from a receptacle.
- put (object) in/on (receptacle): Place the object you are holding into or onto a
↪ receptacle.
- examine (object): Examine an object closely to learn its properties.

## Changing Object States:
- heat (object) with (receptacle): Heat an object with a device (e.g., microwave).
- cool (object) with (receptacle): Cool an object with a device (e.g., fridge).
- clean (object) with (receptacle): Clean an object with a device (e.g., sink).
- slice (object) with (object): Slice an object using a sharp object (e.g., knife).

Important Rules
1. You must first navigate to a location before interacting with objects there
2. You can only hold one object at a time
3. Some objects need to be heated, cooled, or cleaned before placing them
4. Always check admissible commands to see what actions are currently valid

# Checkpoints and Rewind Capability
You operate under two budgets, both shown in each observation message.

The first is a global action budget for the entire episode. Each model action step requests
↪  exactly one environment action or one rewind. This is a hard ceiling shared across
↪ exploration, trial and error, and solving the task. This budget is accumulated globally
↪ and is never refunded by rewinding.

The second is a current-branch budget for the current execution segment. You should aim to
↪ complete the task within the remaining allowance. After a rewind, a new segment starts
↪ from the restored checkpoint with a fresh current-branch allowance. The global action
```

````
↪ budget remains spent. Always use the remaining budgets reported in the latest
↪ observation.

Use these budgets together to balance exploration, learning from failed branches, and
↪ direct progress toward solving the task.

Checkpoints track the current path as C_0, C_1, ..., C_k:
- C_0 is the initial household state.
- C_i is the state reached after i executed environment actions on the current path.

Once you have executed at least one environment action, you may travel back to an earlier
↪ checkpoint listed under Valid rewind targets:
```action
rewind to C_j
```
where 0 <= j < the current checkpoint index.

Rewinding restores the environment to checkpoint C_j, including your location, inventory,
↪ object locations, and object states. Changes made after C_j are undone, but what you
↪ learned on the abandoned branch persists as branch memory. Use this memory to guide
↪ future actions; do not assume that an object remains held, moved, heated, cooled, or
↪ cleaned unless that was already true at C_j.

You can rewind when:
- You find a better plan that starts at an earlier checkpoint.
- The current branch is stuck in repeated invalid actions, no-effect actions, a dead end,
↪ or unproductive exploration.
- You chose the wrong object or receptacle, or performed navigation, open/take/put, or
↪ state-change steps in an unhelpful order.
- The current-branch budget is nearly exhausted and knowledge from this branch can help you
↪  complete the task more efficiently from an earlier checkpoint.

Retain useful progress when choosing a checkpoint. Use learned object locations,
↪ prerequisites, and failed-action outcomes to form an improved plan from the restored
↪ state.

Response Format
Think through your approach step by step, then provide exactly one environment action or
↪ rewind command in the following format:
<think>
Your reasoning and analysis here...
</think>
```action
<your action here>
```

you must adhere to the response format strictly or your action will not be executed:
For example:
<think>
I need to go to the countertop to get the knife.
</think>
```action
go to countertop 1
```

To rewind instead:
<think>
I chose the wrong object. I will return to C_2 and use what I learned to choose the correct
↪  object.
</think>
```action
rewind to C_2
```
Use this rewind example only when C_2 is listed as a valid target.

Always provide exactly ONE action code block at a time, containing either one environment
↪ action or one rewind command. Do not combine them in the same response.
````

ALFWorld Rewind Reflection Prompt

```
You are reflecting on a failed ALFWorld branch.

# Rewind trigger
{rewind_reason}

# Current checkpoint
The branch is currently at C_{current_position}.
```

```
# Valid rewind targets
{valid_targets}

# Branch history
{branch_history}

Write a compact memory that helps the next attempt avoid the failed branch.
Include wrong locations, wrong object choices, missed open/take/put/state-change steps,
and useful facts learned from observations.

Return memory inside <remark>...</remark>, then end with exactly one action block:
```action
rewind to C_j
```
```

ALFWorld Observation-Message Example (user role)

```
Current observation at checkpoint C_1:
{observation}

Admissible commands:
  - {admissible_command}

Global action budget used: 1/60 (remaining: 59).
Current-branch budget used: 1/20 (remaining: 19).
Valid rewind targets: C_0.

Choose exactly one environment action in ```action ... ```, or travel back with ```action
rewind to C_j
```.
```

### E.4 SCIENCEWORLD

Reward variants share the following system and reflection templates. The initial user message includes the task description. Later observations include admissible actions, budgets, branch memories, and the native score.

**ScienceWorld Rewind System Prompt**

```
You are an expert autonomous agent operating in the ScienceWorld environment, a text-based
↪ virtual world for accomplishing tasks from the elementary science curriculum, with
↪ REWIND capability.

You interact by issuing one free-text command per turn. Each turn you are shown the current
↪ observation and the admissible actions (an action template list plus the objects that
↪ can replace OBJ). Typical actions include: look around, open OBJ, close OBJ, activate
↪ OBJ, deactivate OBJ, go to LOC, teleport to LOC, pick up OBJ, move OBJ to OBJ, focus on
↪ OBJ, read OBJ, wait, task.

Important rules:
1. You must navigate to a location before interacting with the objects there.
2. Replace OBJ / LOC placeholders with concrete objects and locations from the observation.
3. `focus on OBJ` signals the object the task is about | use it deliberately, it can end
↪ the task.

Checkpoints and Rewind Capability
You operate under two budgets.

The first is a global action budget: {step_budget} model action steps for the
entire episode. This budget is never refunded by rewinding.

The second is a task-solve budget: {branch_attempt_budget} model action steps
for progress along the current active checkpoint path. When you rewind to
checkpoint C_j, this task-solve budget rolls back to the amount already consumed
at C_j. Knowledge from abandoned branches persists as memory.

Checkpoints track the current path as C_0, C_1, ..., C_k:
- C_0 is the initial environment state.
- C_i is the state reached after i executed actions on the current path.

Once you have executed at least one action, you may travel back to any
earlier checkpoint:
- `<action>rewind to C_j</action>`, where 0 <= j < the current checkpoint index.
- `<action>rewind to j</action>` is equivalent.
```
```

```
Use rewind when a branch reaches a dead end, becomes inconsistent with earlier
observations, repeats ineffective actions, or when a better plan should restart
from an earlier state.

# Response Format
- First reason step-by-step about the task, the observation, and the admissible actions.
- Then give exactly ONE action: `<action>focus on door</action>`.
- OR rewind to a previous checkpoint: `<action>rewind to C_j</action>`.
Only the final `<action>...</action>` tag will be executed.
```

ScienceWorld Rewind Reflection Prompt

```
You are an expert ScienceWorld agent reflecting on a branch attempt.

# Core rollback semantics
The current branch is at C_{rewind_from_step}. The model must choose or confirm
the rewind target before the environment is restored.
The knowledge state must move forward: compress the branch attempt into a small branch
↪ memory.
Do not preserve the full trajectory. Preserve only information useful for future decisions.

{rewind_target_instruction}

# Rewind trigger
{rewind_reason}

# Valid rollback targets
{valid_targets}

# Relevant checkpoint states
{checkpoint_context}

# State at C_{rewind_from_step} before rewinding
{current_observation}

# Branch history from C_{history_start_step} to C_{rewind_from_step}
{history_after_rewind}

# Prior reflection outputs
{reflection_history_context}

# Your task
Reflect on what went wrong and produce a concise branch memory for the next attempt.
Focus on:
- wrong locations, wrong object choices, or ineffective actions that should not be repeated
↪ ;
- missed navigation, open/activate/move/focus, or state-change steps;
- useful facts learned from observations about objects and locations;
- whether the branch should restart before or after a useful action;
- a concrete improved plan from the checkpoint you return to.

Include a compact branch memory inside <remark> </remark> tags.
Use this structure, replacing C_j with the checkpoint you choose or confirm:
<remark>
Failure reason: ...
Useful facts learned: ...
Avoid repeating: ...
Next new plan from C_j: ...
</remark>

End with exactly one final action tag selecting the checkpoint to restore:
<action>rewind to C_j</action>
```

ScienceWorld Observation-Message Example (user role)

```
Current observation at checkpoint C_1 (segment turn 2):
{observation}

Score so far: 0/100.
Admissible actions:
{available_actions}

Global action budget used: 1/60 (remaining: 59).
Task-solve budget used on this branch: 1/20 (remaining: 19).
Current checkpoint: C_1.
Valid model rewind targets: C_0.
```

```
Choose exactly one action inside one final action tag, e.g. <action>focus on door</action>,
↪  or travel back: <action>rewind to C_j</action>.
```

F CASE STUDIES

These cases illustrate failure-triggered recovery in MineSweeper and proactive recovery in Sokoban. Each trace follows the abandoned branch, the retained reflection, and the resumed actions.

Logged cases and protocol. The cases are selected validation rollouts from archived training variants, not runs underlying the final evaluation tables. Their prompts allow 60 action turns globally and 20 on the active path; the main protocol instead counts execution and reflection together within 60 model calls. We therefore report action turns, primitive actions, and reflection calls separately. Checkpoint indices can be reused after rewind. Boxed excerpts preserve the logged output, with omissions marked `[...]`; surrounding text is our analysis. Outcomes were checked against rollout metrics and independent state replay.

F.1 MINESWEEPER: RECOVERING AFTER A FAILED REVEAL

The task is to reveal 33 safe cells on a 6×6 board with three mines. Reveals at `(4,1)`, `(5,1)`, and `(6,1)` produce clues 2, 2, and 1. At C_3, 32 safe cells are open. The policy then reveals `(6,2)`, hits a mine, and ends the branch. It had acknowledged that `(5,2)` was unopened but incorrectly treated `(6,2)` as the only remaining cell.

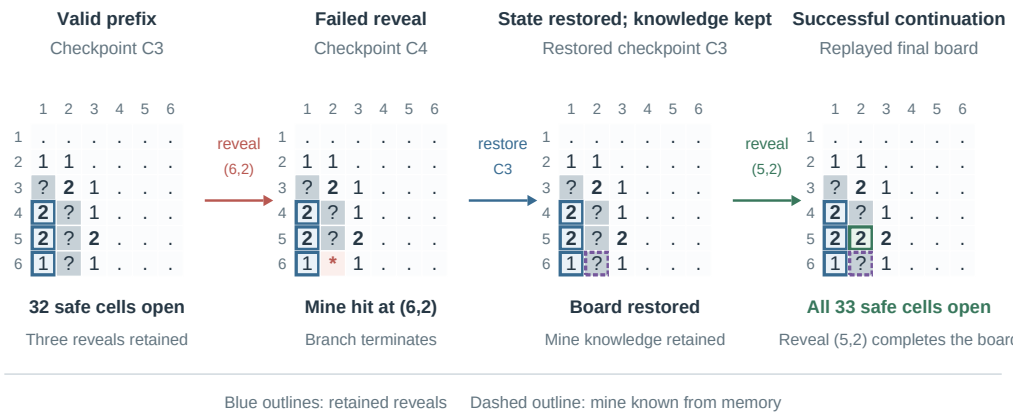

Figure 6: **MineSweeper: restoring the board while retaining the failed reveal's outcome.** Revealing `(6,2)` ends the branch. The policy selects C_3, preserving all three preceding safe reveals, and carries the mine location into the resumed execution. It then reveals `(5,2)`, completing the board. Coordinates are one-indexed.

Observed branch and recovery

Valid prefix, $C_0 \to C_3$ `(4,1)` $\to 2$; `(5,1)` $\to 2$; `(6,1)` $\to 1$.

Erroneous reveal, $C_3 \to C_4$ `(6,2)` $\to$ mine hit; the environment records `done=True, won=False`.

Recovery The failure triggers reflection. The policy selects C_3, restoring the board before the mine hit while preserving the three safe reveals.

Policy reflection and selected checkpoint (excerpts)

```
<remark>
[...]
But (6,2) was revealed and it's a mine — that means (5,2) is safe.
[...]
Revealing (5,2) will complete all reveals.

Final action: <action>(5,2)</action>
```

```
[...]
</remark>

<action>rewind to C_3</action>
```

The next execution prompt contains the entire remark. Although restoration hides `(6,2)` again, the policy remembers the mine and chooses the other cell:

Resumed policy output (excerpts)

```
So (6,2) is a mine [...]
[...]
Therefore, the only cell left to reveal is (5,2).

Revealing (5,2) will complete all safe reveals.

Final action:
<action>(5,2)</action>
```

The final reveal completes the board. The run uses five reveals and one reflection call, without repeating a reveal after restoration. A single-episode rollout on the same instance follows the same four-action prefix and ends at the mine hit.

What this case shows. The board returns to C_3 while the policy retains knowledge from the failed branch and preserves the three safe reveals. The original error was avoidable: the visible clues already determined the mine layout. Although parts of the reflection are redundant or inconsistent about completion, its final mine identification and recommended action agree with the successful continuation.

F.2 Sokoban: Leaving a Loop While Preserving a Subgoal

The task is to place two boxes on targets in a 7×7 room. At C_4, one box is on target at `(4,1)`, the player is at `(4,4)`, and the other box must move from `(3,4)` to `(3,2)`. Four successive `Right`, `Left` commands return the player to the same position without moving either box. Two further `Left` commands also make no box progress. At C_{10}, the policy requests `rewind to C_4` before any terminal failure.

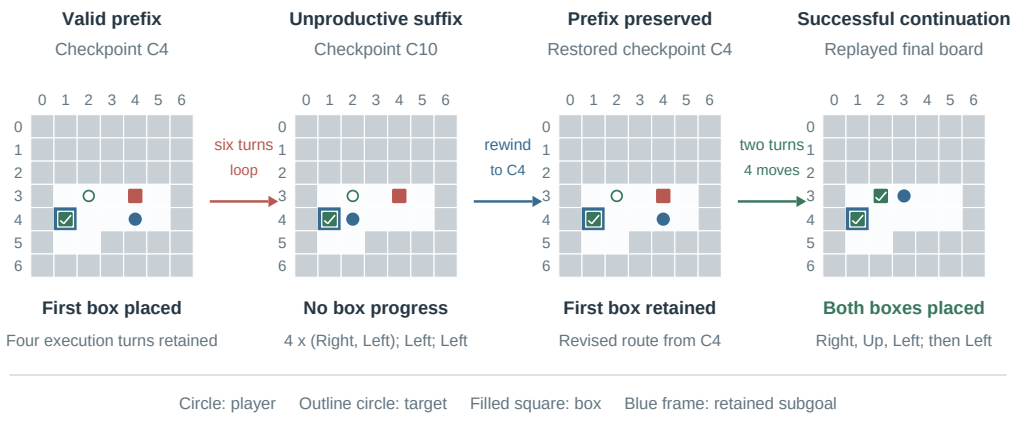

Figure 7: **Sokoban: replacing an unproductive continuation while preserving a solved subgoal.** The policy returns from C_{10} to C_4, retaining the first box on its target. The final reflection identifies how to reach the right side of the remaining box; the next two action turns execute that route and complete the task. Coordinates follow the zero-indexed environment rendering.

Observed branch and model-initiated rewind

Retained prefix, $C_0 \rightarrow C_4$ Four action turns place the first box on `(4,1)` and leave the player at `(4,4)`.

Repeated loop, $C_4 \rightarrow C_8$ Four `Right, Left` commands each return to the same board state; neither box moves.

Further detour, $C_8 \rightarrow C_{10}$ Two `Left` commands move the player to `(4,2)`. Both box positions remain unchanged.

Policy decision at C_{10} `<action>rewind to C_4</action>`. The restored state keeps the first box on target and the four-turn prefix.

After initially misreading the grid, the reflection identifies a route to the right side of the remaining box: move from `(4,4)` through `(4,5)` to `(3,5)`, then push left twice:

Final part of the policy reflection (excerpts)

```
<remark>
[...]
So the correct plan from C_4 must:
- Right to (4,5)
- Up to (3,5)
- Left to push (3,4) to (3,3)
- Left again to push (3,3) to (3,2)
[...]
Next new Plan from C_4: Right, Up, Left, Left
</remark>
```

The runner retains the last 4,800 of the remark's 6,706 characters, prefixed by `[truncated]`. Both quoted plan blocks survive. The next two calls follow the plan:

The next two policy actions (excerpts)

```
<action>right, up, left</action>
[...]
<action>left</action>
```

The calls move the box through `(3,3)` onto `(3,2)`, while the first box stays on target at `(4,1)`. The run uses 13 action turns, including one rewind request, plus one reflection call and 20 primitive movement attempts. No forced rewind occurs.

What this case shows. The policy leaves a loop, preserves a solved subgoal, and follows the retained plan. The pre-rewind room was still solvable. This case does not show an efficiency advantage: a single-episode rollout on the same instance succeeds in three action turns. In both cases, the logs confirm that reflection was supplied and its final recommendation followed; they do not isolate its effect from that of state restoration.