ROLLBACK THE WORLD, KEEP THE REFLECTION: ROLLBACK-INDUCED REFLECTION FOR LONG-HORIZON LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly tackle long-horizon tasks through multi-step environment interaction, yet a single erroneous action can alter subsequent states and observations, causing errors to compound over time. Existing methods either correct the context without repairing altered environment states or restore earlier states while discarding useful experience, making it difficult to both eliminate failure conditions and avoid repeating past mistakes. We argue that reliable recovery should instead be treated as a rollback-boundary control problem that jointly determines *when* to intervene, *where* to resume, and *what* information should survive recovery. Based on this view, we propose Rollback-Induced Reflection (RIR), a unified recovery framework that restores execution to a selected prior state while carrying forward reusable knowledge distilled from the abandoned trajectory to guide subsequent decisions. We further characterize recovery through a unified operator over rollback depth and retained memory, providing a general view of state restoration and knowledge retention. Experiments on three long-horizon benchmarks show that RIR consistently improves average task performance across multiple LLM backbones, with structured reflection memory preserving useful experience and selective rollback enabling efficient recovery.

1 INTRODUCTION

Large language model (LLM) agents can solve complex long-horizon tasks through reasoning, tool use, and environment interaction (Wei et al., 2022; Yao et al., 2022; Park et al., 2023). Yet such tasks are highly sensitive to erroneous actions: a single misstep can alter subsequent states and observations, causing later decisions to rely on corrupted context. Errors therefore compound over time and can drive the agent progressively away from a valid solution trajectory (Hao et al., 2026).

Existing remedies intervene at two levels. *Information-level* methods append corrective feedback to guide future decisions (Madaan et al., 2023; Shinn et al., 2023; Zhao et al., 2024; Kim et al., 2025), but cannot undo environmental changes already caused by an erroneous action, and contaminated observations remaining in context may contradict the corrective advice itself. *State-level* methods instead restore execution to an earlier checkpoint and discard the erroneous suffix (Zhou et al., 2023; Li et al., 2025; Zhang et al., 2026c; Hao et al., 2026), but rewinding the state does not decide which claims from that suffix survive it. Retain too little and the agent may repeat the same failure; overgeneralize a local failure and viable alternatives may be incorrectly ruled out.

A reliable rollback mechanism must therefore answer three coupled questions. First, *when to roll back*: is the current branch still productive exploration, or has it begun to propagate errors? Intervening too early curtails valid exploration; too late lets the consequences of an error compound. Second, *where to roll back*: which checkpoint eliminates the conditions that produced the failure while preserving the most valid progress? Going too far back sacrifices completed work; not far enough leaves the underlying obstacle in place. Third, *what to roll back*: which claims in the discarded suffix are invalidated along with the execution state, and which should survive as knowledge for the next attempt?

Together, these questions define a rollback boundary: *when* decides whether to draw it, *where* places it along the trajectory, and *what* controls which information crosses it. At this boundary, the system

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.

restores the environment and branch-local agent context to the selected checkpoint and removes the subsequent suffix from the active trajectory. The suffix, however, need not be discarded wholesale. State claims invalidated by restoration are removed, while reusable observations and lessons are distilled into reflective knowledge for the next attempt. Rollback thus becomes both a state transition and an opportunity to reconstruct the decision context: **rollback the world, keep the reflection**.

Building on this view, we propose **Rollback-Induced Reflection (RIR)**, a recovery-control framework for long-horizon LLM agents. For *when*, **Hybrid Adaptive Review** combines agent-initiated and adaptive scheduled reviews to assess whether the current branch should continue or recover. For *where*, **Coarse-to-Fine Restore Localization** first narrows the search to a causally relevant interval and then selects a checkpoint that balances failure removal against progress preservation. For *what*, **Rollback-Consistent Reflection Memory** separates branch-local state restored with the checkpoint from reusable knowledge that persists across rollback. The memory stores the stable task objective, reusable environment knowledge, past-attempt milestones, and conditioned failure analysis, while deliberately excluding the agent's current state to avoid reintroducing stale claims after restoration.

Formally, we characterize RIR through a unified recovery operator parameterized by a rollback depth k and the memory $\mathcal{M}^+$ retained across the recovery boundary. We show that common correction and rollback mechanisms are restricted cases of the unified operator, and that their induced recovery-policy classes are therefore contained within the RIR recovery space. This policy-class inclusion leads directly to an optimal-value monotonicity result, under which the best task-completion probability attainable by RIR is no lower than that of any such restricted mechanism. Empirically, RIR consistently outperforms representative baselines across multiple long-horizon benchmarks and LLM backbones, improving average success rate by up to 6.57 percentage points while maintaining selective recovery under constrained interaction budgets. Our contributions are as follows:

- We recast trajectory contamination in long-horizon agent tasks as a problem of *rollback boundary* placement and identify three coupled questions that recovery must resolve: when to intervene, where to resume, and what information should survive the rollback.
- We propose **Rollback-Induced Reflection (RIR)**, which combines adaptive review, coarse-to-fine restore localization, and rollback-consistent reflection to recover execution state without discarding reusable knowledge from failed branches.
- We formalize RIR through a unified recovery operator and establish policy-class containment and optimal-value monotonicity for mechanisms that restrict rollback depth within a shared memory-update space. Experiments on long-horizon benchmarks further validate its effectiveness under constrained interaction budgets.

2 Related Work

LLM agents increasingly use context-management mechanisms such as external memory, experience retrieval, and selective context construction to support long-horizon interaction (Yu et al., 2026; Chhikara et al., 2025; Jia et al., 2026; Lu et al., 2026). However, these methods generally do not address what happens once an erroneous action has already altered the environment: whether and how execution itself should be recovered (Zhang et al., 2026c; Hu et al., 2025; Wu et al., 2025). We therefore focus on rollback-oriented recovery and group prior work into two broad forms.

Information-level correction. Self-Refine (Madaan et al., 2023), AgenTracer (Zhang et al., 2026b), Reflexion (Shinn et al., 2023), and ReflAct (Kim et al., 2025) improve subsequent decisions through feedback or reflection, while ExpeL (Zhao et al., 2024), AutoGuide (Fu et al., 2024), and G-Memory (Zhang et al., 2026a) reuse experience across tasks. Although effective feedback can improve future behavior (Huang et al., 2024; Kamoi et al., 2024), information-level correction cannot undo environmental consequences already caused by erroneous actions, and retained branch-local state claims may become stale after restoration.

State-level rollback. GA-Rollback (Li et al., 2025) and WebRollback (Zhang et al., 2026c) explicitly restore earlier states in interactive trajectories, while SRC (Hao et al., 2026) and DART (Yang et al., 2026) study rollback for training-data construction and structured recoverability, respectively. These methods demonstrate the value of explicitly restoring execution state, but typically focus on failure detection or restore-point selection rather than the full recovery boundary.

In contrast, RIR treats the recovery boundary itself as an explicit control problem: it unifies state restoration and context reconstruction by jointly deciding *when* to recover, *where* to resume, and *what* information remains valid across the rollback boundary. Unlike methods that treat reflection or rollback in isolation, RIR restores execution while preserving reusable knowledge and excluding state claims invalidated by restoration.

3 Problem Formulation

We consider a partially observable long-horizon interactive task $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, P, \Omega, R_g)$, where $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ denote the state, action, and observation spaces. The environment evolves according to $P(s_{t+1}|s_t, a_t)$ and emits observations through $\Omega(o_{t+1}|s_{t+1})$. Given task objective g, the outcome function $R_g(s_T, y_T)$ evaluates the terminal execution between the final state s_T and the ground truth state y_T, it is binary for verifiable tasks and may be real-valued when graded rewards are available.

At step t, the base agent follows an LLM policy $a_t \sim \pi_\theta(\cdot|g, h_t, \mathcal{M})$, where $\mathcal{M}$ is the persistent Reflection Memory and $h_t = (o_0, a_0, \ldots, a_{t-1}, o_t)$ is the interaction history of the current branch. The key distinction is that h_t is checkpointed with execution, whereas $\mathcal{M}$ is maintained outside checkpoints and may carry knowledge across recovery attempts. The action space consists of final responses, ordinary task tools, and a recovery-control tool: $\mathcal{A} = \mathcal{A}_{\text{resp}} \cup \mathcal{A}_{\text{tool}}, \mathcal{A}_{\text{tool}} = \mathcal{A}_{\text{task}} \cup \{\texttt{rollback}\}$. Actions in $\mathcal{A}_{\text{task}}$ interact with the environment, while $\mathcal{A}_{\text{resp}}$ terminates the episode with a final response. The `rollback` tool instead submits a recovery request and does not directly modify the environment.

Rollback. Before each executable action, the system stores a checkpoint $C_i = (s_i, h_i)$, containing the environment state and branch-local context at step i. Let $\mathcal{I}_t = \{\, i \le t \mid C_i \text{ is restorable at step } t \,\}$ denote the set of admissible restore points. For a selected restore point $r \in \mathcal{I}_t$, we define the trajectory discarded by recovery as the *abandoned suffix* $\tau_{r:t} = (a_r, o_{r+1}, \ldots, a_{t-1}, o_t)$.

A rollback first extracts reusable knowledge from $\tau_{r:t}$ to update the Reflection Memory, and then restores the environment and branch-local context to (s_r, h_r). Execution subsequently resumes from the restored checkpoint under the updated memory. We define $k = t - r$ as the *rollback depth*. We allow the limiting case $k = 0$, corresponding to context correction without state restoration.

Objective. RIR leaves the parameters θ of the base agent unchanged and instead introduces a test-time recovery control policy Π. Given an agent-call budget B_{agent} and a rollback budget B_{rb}, RIR seeks to maximize task success under bounded interaction:

$$\max_{\Pi} \ \mathbb{E}_{\pi_\theta, \Pi, \mathcal{E}}[R_g(s_T, y_T)] \quad \text{s.t.} \quad N_{\text{agent}} \le B_{\text{agent}}, \ \ N_{\text{rb}} \le B_{\text{rb}}. \tag{1}$$

4 Rollback-Induced Reflection Framework

4.1 Overview

As shown in Figure 1, RIR organizes recovery around three coupled decisions: **When** decides whether the current branch should continue, **Where** selects the checkpoint to resume from, and **What** determines which information from the abandoned suffix should survive the rollback. Algorithm 1 summarizes the complete execution loop.

During execution, RIR stores checkpoints along the current branch. A review is triggered either by an agent recovery request or by the adaptive review schedule. If the branch is judged promising, the system continues and adjusts the next review interval. Otherwise, RIR first localizes a restore checkpoint coarse-to-fine, then distills reflective knowledge from the suffix about to be abandoned. Finally, the system restores the environment and the agent context while retaining the updated Reflection Memory to guide the new attempt. The procedure can be summarized as:

$$\mathcal{M}^+ = \phi(\mathcal{M}, \tau_{r:t}), \quad (s_t, h_t, \mathcal{M}) \xrightarrow{\text{rollback}} (s_r, h_r, \mathcal{M}^+), \quad r = t - k, \tag{2}$$

where ϕ denotes the information update instantiated in Section 4.4. Together with rollback depth k, the pair $(k, \mathcal{M}^+)$ defines the unified recovery operator analyzed in Section 4.5. Equation 2 makes RIR's central asymmetry explicit: the execution world and branch-local context return to the past, while reflective knowledge acquired from interactions that have already occurred is carried forward.

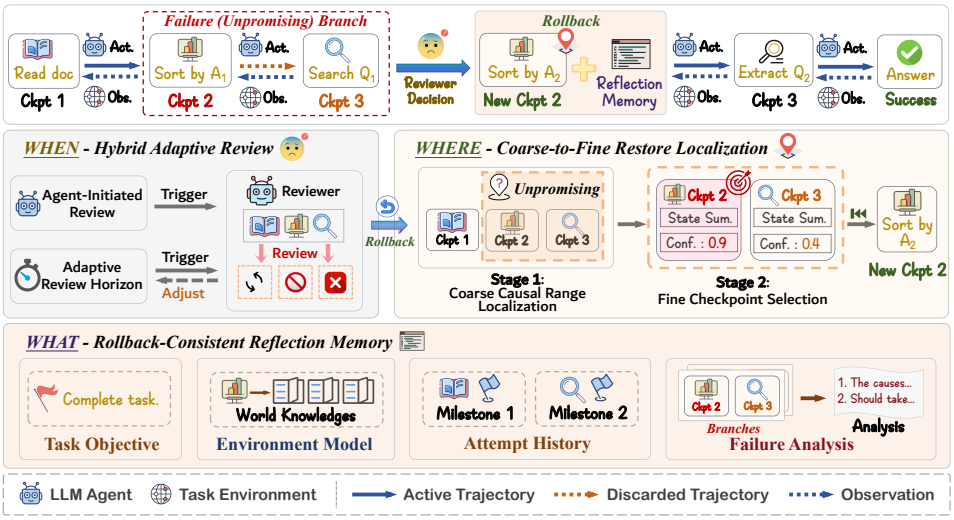


Figure 1: Overview of the Rollback-Induced Reflection (RIR) framework.

4.2 When: Hybrid Adaptive Review

Intervening too early truncates legitimate exploration, whereas intervening too late allows errors to compound. To strike a balance, RIR invokes a single reviewer through two complementary triggers: **agent-initiated review**, where the agent calls `rollback` as a soft recovery request, and **adaptive scheduled review**, where the controller inspects the trajectory after a dynamic review horizon. The former captures failures recognized by the agent itself, while the latter detects stagnation or errors that the agent may overlook. Their combination reduces reliance on either imperfect signal alone: agent requests enable timely intervention, while scheduled reviews provide an external safeguard against unnoticed failure. The rollback tool schema is provided in Appendix A.1.

Let t_{rev} denote the most recent review step and L_t the current review horizon. A review is triggered when:

$$\text{ReviewEvent}_t = \mathbb{I}[a_t = \texttt{rollback}] \ \vee \ \mathbb{I}[t - t_{\text{rev}} \geq L_t]\,. \tag{3}$$

Importantly, calling `rollback` only requests review and does not directly restore the environment. This separation prevents the agent's local uncertainty from being converted immediately into an irreversible recovery decision. When a review is triggered, the LLM-based reviewer evaluates the task objective g, current trajectory h_t, Reflection Memory M, and auxiliary execution signals ξ_t:

$$(\delta_t, \rho_t, f_t, \Delta E_t) = D_\psi(g, h_t, \mathcal{M}, \xi_t), \tag{4}$$

where $\delta_t \in \{\textsc{Continue}, \textsc{Rollback}\}$ is the recovery verdict, $\rho_t \in [0, 1]$ controls the urgency of the next review, f_t is the failure diagnosis used when recovery is selected, and ΔE_t contains newly established environment knowledge. Here, ξ_t summarizes lightweight execution signals such as repeated actions, no-effect steps, and state cycles. If the reviewer elects to continue, it uses ρ_t to set the next review horizon:

$$L_{t+1} = L_{\min} + (1 - \rho_t)\,(L_{\max} - L_{\min}), \tag{5}$$

so higher urgency leads to earlier re-evaluation. This adaptive horizon concentrates review effort on uncertain trajectories while allowing stable branches longer uninterrupted exploration. More details and the instruction prompt of the reviewer are provided in Appendix A.2.

4.3 Where: Coarse-to-Fine Restore Localization

Once recovery is approved, RIR must select a restore point from the stored checkpoints. Identifying the precise erroneous step in a single pass over the full trajectory is unreliable: failures in long-horizon tasks often arise from several interdependent decisions, while adjacent checkpoints may differ only marginally. We therefore adopt a two-stage coarse-to-fine localization procedure.

In the **coarse causal range localization** stage, an LLM-based selector uses the conditioned failure diagnosis f_t together with the executed trajectory to identify a historical interval likely to contain the decisive error or an unmet precondition:

$$[l, u] = S_{\text{coarse}}(g, h_t, \mathcal{M}, f_t), \quad l, u \in \mathcal{I}_t, \;\; l \leq u. \tag{6}$$

Rather than prematurely committing to a single culprit action, this stage eliminates large portions of the trajectory that are unlikely to have contributed to the current failure and restricts the subsequent search to a compact causal range.

In the **fine checkpoint selection** stage, the selector compares compact state summaries $\sigma(C_i)$ within the localized interval and chooses a restore point that removes the current obstruction while preserving as much valid progress as possible:

$$r = S_{\text{fine}}(g, f_t, \mathcal{M}, \{(i, \sigma(C_i)) \mid i \in \mathcal{I}_t,\ l \leq i \leq u\}). \tag{7}$$

Here $\sigma(C_i)$ denotes a compact description derived from the checkpointed state, such as completed sub-goals, current location, and key resources; it is computed from the checkpoint and is not maintained as an additional memory variable. Let $\text{Escape}(\sigma(C_i), f_t)$ indicate whether restoring checkpoint C_i removes the failure condition characterized by f_t. RIR follows the selection principle:

$$r^\star = \max \left\{ i \in [l, u] \cap \mathcal{I}_t \;\middle|\; \text{Escape}(\sigma(C_i), f_t) = 1 \right\}. \tag{8}$$

In other words, RIR first filters out checkpoints whose restored states still retain the obstruction identified in f_t, and then selects the *latest* checkpoint among the remaining candidates. The first criterion avoids immediately recreating the same failure condition after restoration, while the second preserves the longest valid prefix of the executed trajectory.

In practice, S_{fine} approximates Equation 8 by jointly comparing the candidate state summaries against the conditioned failure diagnosis, and returns a restore point together with its justification and confidence score. The coarse stage identifies *which region of the trajectory is causally relevant to the failure*, whereas the fine stage determines *which restorable state within that region provides the best point of re-entry*. This coarse-to-fine decomposition avoids reducing restore localization to single-step error attribution and instead balances two competing objectives: removing the conditions that caused the failure and preserving as much correct progress as possible. The detailed selector prompts are provided in Appendix A.3.

4.4 WHAT: ROLLBACK-CONSISTENT REFLECTION MEMORY

State restoration determines which trajectory suffix is removed, but not what information from that suffix should survive. Discarding it entirely loses reusable experience, whereas retaining it verbatim may preserve branch-local state claims that rollback has already invalidated. RIR therefore maintains a structured Reflection Memory $\mathcal{M}$ outside the checkpoint:

$$\mathcal{M} = (G, E, H, F). \tag{9}$$

Task Objective (G). Stores a rollback-invariant representation of the final objective and provides stable task-level guidance to the reviewer and the selector.

Environment Model (E). Stores reusable knowledge about the environment across observed branches. We organize it into three semantic channels $E = E^{\text{obs}} \cup E^{\text{elim}} \cup E^{\text{aff}}$, corresponding respectively to observed facts, evidence-supported eliminations, and action or tool affordances. Entries are organized through constrained structured channels intended to reduce the risk of treating information from an abandoned branch as current state. Observations that depend on actions subsequently undone by rollback should be treated as historical evidence requiring re-verification, rather than as facts about the restored world.

Attempt History (H). Records milestones and routes achieved in previous attempts as historical facts rather than claims about the current state. A later attempt can therefore reuse a discovered route without assuming that resources, locations, or progress obtained before rollback remain valid after restoration. The currently executing branch remains represented by h_t; its verified milestones are incorporated into H only when that attempt terminates or is abandoned.

Failure Analysis (F). Summarizes why the most recently abandoned branch failed under the conditions that were in force at the time. Failure patterns that become supported across attempts are

Algorithm 1 Rollback-Induced Reflection (RIR)

Require: Task objective g, base agent π_θ, rollback budget B_{rb}

Initialize Reflection Memory $\mathcal{M}$, review horizon $L \leftarrow L_0$, and $N_{\mathrm{rb}} \leftarrow 0$
while task not terminated **do**
  Store checkpoint $C_t = (s_t, h_t)$
  Sample $a_t \sim \pi_\theta(\cdot \mid g, h_t, \mathcal{M})$
  if a_t is a final response **then**
    return a_t
  if $a_t =$ `rollback` **or** review horizon is reached **then** ▷ WHEN · review the current branch
    Reviewer evaluation $(\delta_t, \rho_t, f_t, \Delta E_t) \leftarrow \mathcal{D}_\psi(g, h_t, \mathcal{M}, \xi_t)$
    Merge ΔE_t into the Environment Model
    if $\delta_t =$ ROLLBACK **and** $N_{\mathrm{rb}} < B_{\mathrm{rb}}$ **then**
      $[l, u] \leftarrow \mathcal{S}_{\mathrm{coarse}}(g, h_t, \mathcal{M}, f_t)$ ▷ WHERE · localize a causal restore range
      Select restore point $r \leftarrow \mathcal{S}_{\mathrm{fine}}(g, f_t, \mathcal{M}, \{(i, \sigma(C_i)) \mid l \le i \le u\})$
      Update $\mathcal{M}^+ \leftarrow \phi(\mathcal{M}, \tau_{r:t})$ ▷ WHAT · retain reusable knowledge
      Restore the environment and branch-local context to C_r
      Reset the review horizon and increment N_{rb}
      continue
    else
      Adapt the next review horizon using ρ_t
  if $a_t \neq$ `rollback` **then**
    Execute a_t and update the trajectory

instead promoted to the Environment Model. Thus, F explains *why the latest branch failed under its specific conditions*, whereas E captures *what is reusable about how the environment behaves*.

Crucially, none of these fields represents the agent's current branch-local state; that information is supplied by the active trajectory h_t and the restored checkpoint (s_r, h_r). This separation prevents stale state claims from crossing the rollback boundary. At each review, the reviewer extracts newly established environment knowledge ΔE_t. If rollback is executed, the reflection patcher additionally examines the abandoned suffix $\tau_{r:t}$ to extract historical milestones ΔH_t and construct a new failure analysis F_{new}. The resulting update is:

$$E^+ = E \cup \Delta E_t, \quad H^+ = H \cup \Delta H_t, \quad F^+ = F_{\mathrm{new}}, \quad \mathcal{M}^+ = (G, E^+, H^+, F^+). \tag{10}$$

Thus, G remains fixed, E and H accumulate reusable knowledge and history, while F is replaced after each failed attempt. We denote this field-specific update by ϕ, which is applied in Equation 2 to produce the updated memory $\mathcal{M}^+$.

The updated memory $\mathcal{M}^+$ replaces the previous Reflection Memory block in the system prompt rather than being appended to it. Because $\mathcal{M}$ is maintained outside checkpoints, restoring C_r reverts the environment and branch-local context while preserving the updated memory for the next attempt. This realizes the central asymmetry of RIR: **rollback the world, keep the reflection**.

4.5 UNIFIED RECOVERY-SPACE CHARACTERIZATION

Suppose recovery is triggered at step t, with restore point $r = t - k \in \mathcal{I}_t$. Let $\mathbb{M}(\mathcal{M}, \tau_{r:t})$ denote the set of memory configurations reachable by updating $\mathcal{M}$ from the abandoned suffix $\tau_{r:t}$, including the unchanged memory $\mathcal{M}$ and the full update $\phi(\mathcal{M}, \tau_{r:t})$. The unified recovery operator is

$$\mathcal{R}_{k,\mathcal{M}^+} : (s_t, h_t, \mathcal{M}) \mapsto (s_r, h_r, \mathcal{M}^+), \quad k \in \mathcal{K}_t,\ \mathcal{M}^+ \in \mathbb{M}(\mathcal{M}, \tau_{r:t}). \tag{11}$$

RIR instantiates this operator through its restore selector and memory updater ϕ, yielding Equation 2. Since the abandoned suffix depends on the rollback depth, the joint recovery space is $\mathcal{C}_t = \{(k, \mathcal{M}^+) \mid k \in \mathcal{K}_t,\ \mathcal{M}^+ \in \mathbb{M}(\mathcal{M}, \tau_{t-k:t})\}$.

Proposition 1 (Representational Containment). Consider a recovery mechanism j that shares the same admissible memory configurations at each rollback depth but limits its depth choices to $\mathcal{K}_{j,t} \subseteq \mathcal{K}_t$. Its configuration space is therefore $S_{j,t} = \{(k, \mathcal{M}^+) \in \mathcal{C}_t \mid k \in \mathcal{K}_{j,t}\} \subseteq \mathcal{C}_t$. This restriction affects only the available rollback depths; memory retention remains unrestricted within the shared

memory family. Examples include restart-based recovery with $k = t$, fixed-depth rollback, and bounded-depth rollback, whenever the corresponding depths are admissible.

Let Π_{RIR} denote the idealized policy class with access to all configurations in $\mathcal{C}_t$ at each recovery event, and let Π_j denote the class restricted to $S_{j,t}$. With identical base agents, available information, recovery opportunities, and budget-accounting rules, the unrestricted class can reproduce every decision available to the restricted class. Thus, $\Pi_j \subseteq \Pi_{\mathrm{RIR}}$.

This inclusion also orders the optimal values of the two policy classes. For verifiable tasks, let $R_g \in \{0, 1\}$ indicate task success, $J(\pi) = \Pr_\pi(R_g = 1)$ denote the task-completion probability of policy π, and $V^\star(\Pi) = \sup_{\pi\in\Pi} J(\pi)$ denote the optimal value over a policy class Π.

Corollary 1 (Optimal-Value Monotonicity). For a restricted mechanism satisfying Proposition 1,

$$V^\star(\Pi_{\mathrm{RIR}}) \geq V^\star(\Pi_j). \tag{12}$$

The inequality follows by taking the supremum over the nested policy classes and also holds for expected return. Full statements are provided in Appendix B.

5 Experiments

5.1 Experimental Setup

Datasets & metrics. We evaluate RIR on three long-horizon agent benchmarks: ALFWorld (Shridhar et al., 2020), ScienceWorld (Wang et al., 2022), and GAIA (Mialon et al., 2024), covering embodied interaction, scientific experimentation, and open-domain tool use, respectively. We report success rate (SR) on ALFWorld and ScienceWorld, and use an LLM-as-a-Judge evaluator on GAIA to assess semantic equivalence between predicted and reference answers. On ScienceWorld, we additionally report normalized dense reward (DR) to measure partial task progress.

Baselines. We compare against representative training-free test-time methods under the same LLM backbones and interaction budgets: ReAct (Yao et al., 2022) without explicit recovery; Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) for information-level correction; LATS (Zhou et al., 2023) for search-based recovery; and GA-Rollback (Li et al., 2025) for explicit state rollback. We use Qwen3-14B (Yang et al., 2025), DeepSeek-V3 (Liu et al., 2024) and Qwen3.8-27B as backbone models and follow official implementations and recommended configurations whenever available. Further baseline details are provided in Appendix C.3.

Implementation details. We implement the base agent with AgentScope (Gao et al., 2025). Within each backbone setting, the acting agent and all RIR components use the same LLM, so recovery gains cannot be attributed to a stronger auxiliary model. All methods share identical task interfaces and interaction budgets. For GAIA, we additionally restore the workspace state during rollback, including files created or modified by the agent, to keep the external environment consistent with the recovered trajectory. Separately, final-answer evaluation on GAIA uses an independent Qwen3.8-Max judge that does not participate in trajectory generation or recovery. Additional implementation details are provided in Appendix C.4.

5.2 Main Results

Overall performance. As shown in Table 1, RIR achieves the highest average SR under all three backbones and the highest SR in eight of the nine backbone–benchmark settings. With Qwen3-14B, DeepSeek-V3, and Qwen3.8-27B, RIR reaches average success rates of 47.00%, 69.43%, and 76.54%, outperforming the strongest corresponding baseline by 3.09, 6.57, and 3.04 percentage points, respectively. RIR also obtains the highest ScienceWorld DR under all three backbones. The only exception is GAIA with Qwen3.8-27B, where Reflexion leads by a narrow margin of 1.97 percentage points. We attribute this in part to the experimental setting of a 24-step interaction budget, which leaves limited room for RIR to realize the benefits of rollback through post-recovery exploration. Despite this isolated case, RIR's consistent average gains across backbones of varying capability suggest that explicit recovery remains useful rather than being subsumed by improvements in base-model capability. This empirical pattern is consistent with our recovery-space analysis: the broader recovery space provides additional useful choices in practice, although the theoretical result itself is a capacity statement rather than a performance guarantee.

Table 1: Main results across three benchmarks. SR denotes success rate and DR denotes normalized dense reward; Avg. SR is the unweighted mean across the three benchmarks. **Bold** and underlined denote the best and second-best results within each backbone, respectively.

Backbone	Method	ALFWorld	ScienceWorld		GAIA	Avg. SR (%)
		SR(%) ↑	SR(%) ↑	DR ↑	SR(%) ↑	
Qwen3-14B	ReAct	67.16	31.37	0.412	19.08	39.20
	Reflexion	74.63	35.38	0.426	21.71	43.91
	LATS	67.91	29.15	0.407	15.79	37.62
	Self-Refine	66.42	21.03	0.402	15.13	34.19
	GA-Rollback	77.61	26.94	0.387	18.42	40.99
	RIR (Ours)	**81.30**	**36.67**	**0.448**	**23.03**	**47.00**
DeepSeek-V3	ReAct	85.82	67.89	0.779	34.87	62.86
	Reflexion	77.61	62.36	0.735	34.21	58.06
	LATS	61.65	65.31	0.758	32.89	53.28
	Self-Refine	80.60	51.66	0.535	34.87	55.71
	GA-Rollback	62.69	53.51	0.639	26.97	47.72
	RIR (Ours)	**94.03**	**76.75**	**0.813**	**37.50**	**69.43**
Qwen3.8-27B	ReAct	96.27	70.11	0.813	49.34	71.91
	Reflexion	97.01	70.85	0.806	**52.63**	73.50
	LATS	90.30	64.21	0.769	47.37	67.29
	Self-Refine	91.04	74.91	0.832	48.68	71.54
	GA-Rollback	89.55	72.32	0.786	48.03	69.97
	RIR (Ours)	**99.25**	**79.70**	**0.872**	50.66	**76.54**

Efficiency. Figure 2 compares test-time overhead on ScienceWorld across three backbones. RIR remains in the low-overhead regime while requiring substantially fewer auxiliary LLM calls than the more expensive recovery baselines, and this pattern is consistent across Qwen3-14B, DeepSeek-V3 and Qwen3.8-27B. Recovery is also selective rather than frequent: Table 2 shows fewer than 1.3 executed rollback per task for RIR under all backbones, compared with approximately 3.0 for GA-Rollback. Thus, the performance gains are associated with targeted correction rather than repeated rollback and re-exploration. Complete efficiency results are provided in Appendix D.1.

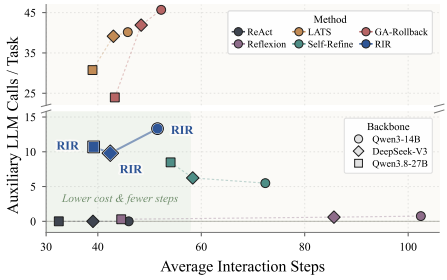


Figure 2: Efficiency trade-off on ScienceWorld across all three backbones.

To complement the aggregate results, we further inspect representative successful and unsuccessful recovery trajectories in Appendix E. The cases illustrate both when rollback provides state-level benefits that reflection alone cannot recover and when incorrect diagnosis can cause recovery to revisit an ineffective branch.

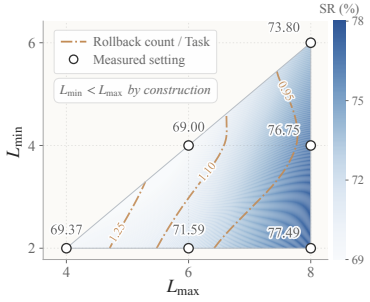


Figure 3: Sensitivity to the adaptive review range $(L_{\min}, L_{\max})$ on ScienceWorld with DeepSeek-V3. Color denotes SR and orange contours denote average rollback frequency.

Table 2: Average executed rollbacks per task on ScienceWorld. Methods without state-level rollback are omitted.

Method	Q.3-14B	D.S.-V3	Q.3.8-27B
GA-Rollback	2.908	2.910	3.112
RIR	**0.978**	**0.920**	**1.266**

5.3 Ablation Studies

When to review. We compare agent-only, scheduled-only, and hybrid review under fixed or adaptive scheduling. As shown in Table 3, the full hybrid-adaptive RIR achieves the highest task performance (76.75% SR, 0.813 DR). Importantly, Hybrid (fixed) performs worse despite triggering more reviews and rollbacks and using more auxiliary LLM calls. Adaptive scheduling therefore appears to improve *when* the two review signals intervene, rather than simply increasing intervention frequency. This supports the design of review timing as an adaptive control decision rather than a fixed periodic mechanism.

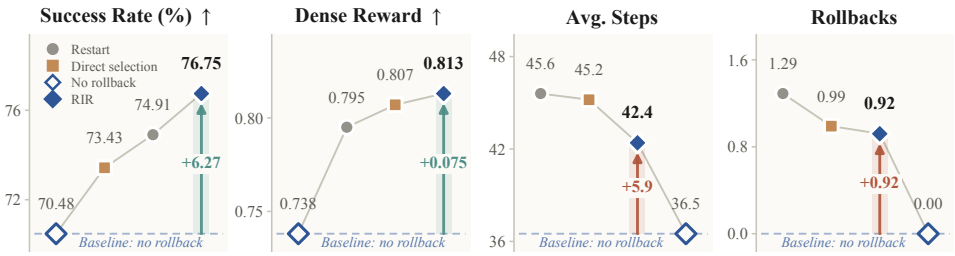


Figure 4: Where-to-recover ablation on ScienceWorld with DeepSeek-V3. *No rollback* retains reflection but disables state restoration; *Restart* restores to the initial checkpoint; *Direct selection* chooses a checkpoint in one stage; and *RIR* uses coarse-to-fine restore localization.

Where to recover. We next vary restore-point localization while keeping the review policy and Reflection Memory unchanged. We compare *No rollback* (reflection only), *Restart*, *Direct selection*, and the full coarse-to-fine RIR selector. Figure 4 shows that No rollback produces the shortest trajectories but substantially lower task performance, indicating that information-level correction alone cannot replace state recovery. Among rollback-based variants, RIR achieves the highest SR (76.75%) and DR (0.813) while requiring the fewest steps (42.4) and rollbacks (0.92).

Rollback vs. Reflection. We further isolate the two core recovery components by comparing *Reflection only*, *Rollback only*, and the full RIR. Table 3 shows that neither component alone matches the complete method: RIR improves SR by 6.27 points over Reflection only and by 7.38 points over Rollback only. Reflection only is cheaper but cannot repair execution-state errors, whereas Rollback only incurs greater interaction and inference cost without preserving reusable knowledge across attempts. In terms of our recovery operator, these variants separately restrict either state restoration or the memory update, while RIR couples both dimensions. The resulting gap therefore empirically supports their complementarity.

Table 3: Ablations on ScienceWorld with DeepSeek-V3 as the backbone. **Avg. Steps**, **Rev.**, **RB**, and **Aux.** report the per-task averages of interaction steps, reviewer invocations, executed rollbacks, and auxiliary LLM calls, respectively.

Variant	SR↑	DR↑	Avg. Steps	Rev.	RB	Aux.
Review-trigger variants						
Agent-only	73.43	0.800	44.2	3.68	0.86	6.28
Sched.-only (fix.)	74.54	0.795	39.2	6.05	0.51	7.58
Sched.-only (adap.)	72.32	0.788	39.0	6.13	0.54	7.76
Hybrid (fixed)	69.00	0.761	46.6	7.79	1.17	11.32
Recovery-component variants						
Reflection only	70.48	0.738	36.5	7.25	/	8.12
Rollback only	69.37	0.801	46.7	7.81	1.30	11.82
RIR	**76.75**	**0.813**	42.4	7.08	0.92	9.85

Dynamic range setting. We finally vary the adaptive review range $(L_{\min}, L_{\max})$. Figure 3 shows that increasing $L_{\max}$ generally improves performance, whereas an overly large $L_{\min}$ is detrimental. The best setting, $(2, 8)$, reaches 77.49% SR with only 0.77 rollbacks per task; the default $(4, 8)$ remains close at 76.75%, indicating that performance is not overly sensitive to a single optimum. Overall, a wider range gives the reviewer more flexibility to intervene quickly on risky branches while allowing stable branches to proceed with fewer interruptions.

6 Conclusion

In this work, we propose Rollback-Induced Reflection (RIR), a unified recovery framework for long-horizon LLM agents. RIR couples adaptive review, restore localization, and persistent reflection to address a key limitation of existing recovery methods: correcting execution state without losing useful experience from failed trajectories. We further characterize recovery through a unified operator over rollback depth and updated reflection memory, showing that several common correction and rollback mechanisms arise as restricted cases of the RIR recovery space and establishing the corresponding optimal-value monotonicity result. Empirically, RIR improves task completion without relying on frequent intervention, while the ablation results show that review timing, restore localization, and reflection play complementary roles in successful recovery. More broadly, our results suggest that reliable long-horizon agents require recovery mechanisms that do more than undo mistakes: they must transform failed interaction into a better basis for subsequent decision-making.

AI USE STATEMENT

In this work, we used generative AI tools for language polishing and grammatical correction, as well as assistance with figure visualization and visualization-code refinement. We have reviewed all AI-assisted work and manually verified the resulting text, visualization code, and figures before incorporating them into the manuscript. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies recovery mechanisms for long-horizon LLM agents and evaluates them on established agent benchmarks. The study does not involve human subjects or private user data. While improved recovery mechanisms can enhance the reliability of autonomous agents, such systems may still inherit limitations and potential risks from their underlying language models, tools, and execution environments. We therefore encourage careful evaluation and appropriate human oversight when deploying such agents in safety-critical or high-stakes settings.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the RIR framework, prompts, theoretical analysis, experimental settings, baseline implementations, evaluation protocols, and hyperparameters in the main paper and appendices. Appendix A provides the prompts and recovery components, Appendix B contains the complete theoretical analysis and proofs, and Appendix C documents the benchmark settings, evaluation protocols, and implementation details. Additional efficiency results and qualitative case studies are provided in Appendices D and E, respectively. An anonymous code implementation is available at `https://anonymous.4open.science/r/anonymous_rir`.

REFERENCES

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

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 119919–119948, 2024.

Dawei Gao, Zitao Li, Yuexiang Xie, Weirui Kuang, Liuyi Yao, Bingchen Qian, Zhijian Ma, Yue Cui, Haohao Luo, Shen Li, et al. Agentscope 1.0: A developer-centric framework for building agentic applications. *arXiv preprint arXiv:2508.16279*, 2025.

Longkun Hao, Hongyu Lin, Hao Li, Zhichao Yang, Haojie Hao, Dongshuo Huang, Haitao Yang, Hongyu Ge, Yanjun Wu, Zi Hao Yin, et al. Speculative rollback correction for quality-diverse web agent imitation. *arXiv preprint arXiv:2606.12485*, 2026.

Minda Hu, Tianqing Fang, Jianshu Zhang, Jun-Yu Ma, Zhisong Zhang, Jingyan Zhou, Hongming Zhang, Haitao Mi, Dong Yu, and Irwin King. Webcot: Enhancing web agent reasoning by reconstructing chain-of-thought in reflection, branching, and rollback. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 5155–5173, 2025.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, volume 2024, pp. 32808–32824, 2024.

Runsong Jia, Mengjia Wu, Ying Ding, Jie Lu, and Yi Zhang. Agent-enhanced heterogeneous graph rag for academic question answering. In *Proceedings of the ACM Web Conference*, pp. 8765–8768, 2026.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024.

Jeonghye Kim, Sojeong Rhee, Minbeom Kim, Dohyung Kim, Sangmook Lee, Youngchul Sung, and Kyomin Jung. Reflact: World-grounded decision making in llm agents via goal-state reflection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33421–33453, 2025.

Xingzuo Li, Kehai Chen, Yunfei Long, Xuefeng Bai, Yong Xu, and Min Zhang. Generator-assistant stepwise rollback framework for large language model agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17694–17711, 2025.

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

Miao Lu, Weiwei Sun, Weihua Du, Zhan Ling, Xuesong Yao, Kang Liu, and Jiecao Chen. Beyond the context window: Scaling agentic rl via end-to-end optimized context compression. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 21074–21125, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pp. 9025–9049, 2024.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

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, pp. 8634–8652, 2023.

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

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Qinzhuo Wu, Pengzhi Gao, Wei Liu, and Jian Luan. Backtrackagent: Enhancing gui agent with error detection and backtracking mechanism. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 4250–4272, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Ke Yang, Panpan Li, Zonghan Wu, Kejin Xu, Huaxi Huang, and Xiaoshui Huang. Dart: Semantic recoverability for structured tool agents. *arXiv preprint arXiv:2605.23311*, 2026.

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

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 21457–21483, 2026.

Guibin Zhang, Muxin Fu, Kun Wang, Frank Wan, Miao Yu, and Shuicheng Yan. G-memory: Tracing hierarchical memory for multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 38, pp. 12988–13018, 2026a.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems? In *International Conference on Learning Representations*, volume 2026, pp. 11377–11399, 2026b.

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*, pp. 187–197, 2026c.

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

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. *arXiv preprint arXiv:2310.04406*, 2023.

A Prompt Design

A.1 The Rollback Tool

RIR exposes exactly one control tool to the base agent. Its interface is deliberately minimal: a single free-text `reason` field, no target step, and no rollback depth. The agent reports that it is stuck and why; it does not choose where the trajectory is restored, which is decided by the reviewer and the selector. The tool is declared read-only, so invoking it cannot alter the task environment: a request costs one step and leaves the world untouched. Figure 5 shows the schema the model receives and the accompanying prompt instruction.

Soft-request semantics. A request is a report, not a command, and the controller may decline it. The three possible outcomes are all returned to the agent as the tool result, so the agent always learns how its request was handled:

- **Granted.** The tool result reads "Rollback request submitted; the controller decides whether to rewind," and control passes to restore localization.
- **Declined by the Reviewer.** The request opens a review with the event type ROLLBACK-REQUEST. If the verdict is CONTINUE, the agent receives "Rollback request rejected by the reviewer," followed by the reviewer's reason, and resumes on the same branch.
- **Declined before review.** If the current attempt has committed no environment action since the last rollback, the request is rejected without consulting the reviewer: restoring an attempt that changed nothing would reproduce the same state. The agent is told "this attempt has taken no action since the last rollback, so rolling back cannot change anything."

Prompt layering. The rollback instruction is an addendum, appended to the system prompt only when the rollback capability is enabled. An agent configured without recovery never receives it and is never shown the tool, so the no-rollback baselines run on the same prompt as the base agent, with no mention of a mechanism they cannot use.

`Rollback` tool schema

```
{
  "name": "Rollback",
  "description":
    "Request that the controller roll back to an earlier checkpoint and revise the
     reflection before continuing.",
  "input_schema": {
    "type": "object",
    "properties": {
      "reason": {
        "type": "string",
        "description":
          "Why the current trajectory should be rolled back."
      }
    },
    "required": ["reason"]
  },
  "is_read_only": true
}
```

Rollback instruction in the system prompt

```
You also have a Rollback tool.  Call it when the current approach is no longer worth
continuing --- for example when recent actions repeat without progress, contradict earlier
verified findings, or rest on an assumption you now believe is wrong.  Rolling back returns
you to an earlier state carrying a lesson about what went wrong, which is usually cheaper
than patching a bad trajectory.  Always give a concrete reason when you call it.
```

Appended to every step:

```
If the current path should be abandoned instead, call the Rollback tool with a concrete
reason.
```

Figure 5: The `rollback` control tool, including the schema exposed to the agent and the system-prompt instruction governing its use. The tool takes only a reason and is read-only, so a request cannot directly modify the environment.

A.2 The Review Process

A review is invoked either when the scheduled horizon is reached or when the agent requests a rollback. The event type is included in the prompt, so the reviewer knows whether it is answering a periodic inspection or an explicit recovery request. In addition to the task objective, the current trajectory, and the Reflection Memory, the prompt supplies the executor's reported state and a small set of run-history counters referenced by the rules below. Figure 6 reproduces the role and rule sections of the reviewer prompt.

The reviewer outputs:

$$(\delta_t, \rho_t, f_t, \Delta E_t), \tag{13}$$

where $\delta_t \in \{\textsc{Continue}, \textsc{Rollback}\}$ is the recovery decision, ρ_t controls the next review horizon, f_t summarizes the current obstruction, and ΔE_t contains environment knowledge established by the reviewed trajectory. These outputs serve different purposes: δ_t determines whether recovery is entered, ρ_t regulates when the branch should next be inspected, and $(f_t, \Delta E_t)$ provide causal and environmental evidence for subsequent recovery and execution.

Evidence-grounded review. The reviewer is deliberately constrained to distinguish observed interaction from the agent's own reasoning. Only executed actions and their returned observations can establish environment knowledge; plans, hypotheses, or mentioned objects alone do not. This prevents speculative reasoning from being promoted into persistent Reflection Memory. The reviewer similarly records only information that is not already represented in the Environment Model, so repeated reviews do not accumulate paraphrases of previously established facts.

Local obstruction versus branch failure. A central behavior of the reviewer is to distinguish a locally repairable action failure from a trajectory-level failure that warrants recovery. An ineffective action does not immediately imply rollback: it may simply reveal an unmet precondition, such as an incorrect agent location, missing preparation step, or invalid target state. In such cases, the reviewer favors CONTINUE and records the observed affordance so that the executor can repair the

Reviewer prompt

```
<role>
You review an agent's current attempt at one task and decide whether it should continue or
be rolled back to an earlier point, and you extend the environment model with what this
review newly established.  You never act in the world yourself.  The environment model is
the only memory you write; attempt history and the failure analysis belong to the rollback
patcher.
</role>
<rules>
Decide in this order; apply the first rule that fits.
1.  Evidence boundary.  Only an executed action - a payload with its returned observation -
  interacted with the world.  Reasoning text, plans, and steps that executed nothing
  interact with nothing; subjects named only in reasoning were not visited, queried, or
  examined.
2.  Report only what is new.  Write findings, eliminations and affordances THIS review
  established and the environment model still lacks.  Do not restate entries the model
  already holds:  the system tracks whether an attempt is still producing knowledge, and
  restating old entries destroys that signal.  Establishing nothing new is a normal outcome
  - return empty lists.
3.  Precondition check - before blaming anything.  When an executed action repeats with no
  effect, the likeliest cause is an unmet precondition:  the agent's own position or state,
  a required prior step, or an unprepared target.  Read executor_state:  it reports what the
  agent's own state is, and a blocked action is very often blocked by it.  That is
  knowledge about the environment's behaviour, not about the task:  record the demonstrated
  rule as an affordance and CONTINUE so the precondition can be satisfied.  Only when the
  precondition hypothesis has been tested and refuted may the repeat count as a futility
  signal.
4.  Futility - the rollback case.  ROLLBACK when returning to an earlier state is what
  unblocks the goal:  (a) the same action still repeating with no new observation after its
  preconditions were verified; (b) a cycle over already-settled subjects; (c) actions
  contradicting the environment model; (d) a plan that provably cannot reach the goal; (e)
  continued work after the needed evidence was already established; (f) the attempt has
  stopped learning - it keeps acting while establishing nothing the environment model
  lacked.  Read the counters:  a high actions_without_effect or revisited_state_ratio is that
  state measured directly, and it does not become acceptable merely because some candidate
  somewhere remains unexamined.
  The test is whether coverage is still EXPANDING, not whether the search is complete.  A
  search is almost never complete, so treating incompleteness as a reason to keep going
  means never judging an attempt at all.  An attempt whose recent actions returned outcomes
  the model already held is orbiting, however many candidates remain in principle.
5.  Rollback cannot fix a belief.  A rollback rewinds decisions, not knowledge - the
  environment model survives it.  If the problem is that the agent believes something the
  model contradicts, rolling back changes nothing; CONTINUE.
6.  Impossibility and completion audit.  Before agreeing that something is impossible,
  check the model:  any goal-relevant positive finding, any incomplete examination, or any
  plausible candidate with no entry refutes the claim - unsupported impossibility warrants
  CONTINUE. Never declare the task complete:  completion is judged only by the environment.
  When everything looks settled, re-read task_objective and keep working on whatever
  required transformation, property, or qualifier is not yet evidence-backed.
7.  Default:  CONTINUE. A NEW observation is progress even before the objective is met, and
  expanding coverage is normal work.  Give a fresh attempt (low actions_since_rollback)
  extra patience.  This default does not survive rule 4(f):  ''it might still find it'' is
  not a new observation.
review_urgency is the only scheduling signal:  1 = review again soon, 0 = allow a long
stretch.
</rules>
```

Figure 6: The reviewer instruction in the system prompt.

precondition within the current branch. Rollback is reserved for cases where the trajectory itself has become unproductive—for example, when the agent cycles over settled states, repeatedly violates established environment knowledge, or follows a plan whose earlier decisions prevent further progress.

This distinction is important because recovery is useful only when changing the execution state can remove the diagnosed obstruction. If the problem is purely informational—for example, the current belief conflicts with evidence already stored in Reflection Memory—restoring an earlier state would not remove that belief. The reviewer therefore favors continued execution in such cases and allows the persistent memory to correct subsequent decisions. Conversely, when the obstruction is tied

Stage 1: Coarse causal range localization

```
<role>
You propose a NARROW interval of checkpoints worth restoring to, given a failed attempt:
from the earliest point that plausibly precedes the failure's root cause, to the latest
point that still escapes it.  Checkpoint i is the state immediately before action i.  You
do not pick the exact checkpoint yet.  The task memory is read-only context.
</role>

<rules>
1.  Only an action payload with its returned observation shows what interacted with the
  world; a reasoning-only step may waste compute, but the subjects in its text were not
  acted on.
2.  The interval's end is the latest checkpoint that still precedes the earliest committing
  step of the failing course - restoring later would keep the mistake.  The interval's
  start is the earliest checkpoint whose extra rewind still buys materially different
  context - rewinding further only discards sound work.
3.  Keep it narrow.  An interval spanning most of the trajectory carries no information and
  will be trimmed by the system; if you cannot justify a narrow interval from the cited
  steps, you have not located the root cause.  State the step indices your bounds rest on.
4.  Return an inclusive interval of checkpoint indices drawn from valid_checkpoint_indices.
</rules>
```

Stage 2: Fine checkpoint selection

```
<role>
You pick the exact checkpoint to restore, strictly from the validated candidate set
(already limited to the justified interval, minus proven dead-ends).  Checkpoint i is the
state immediately before action i.  The task memory is read-only context.
</role>

<rules>
1.  Restoring a checkpoint restores the executor's own state along with the world.  Each
  candidate reports that state.  If the attempt was blocked by an unmet precondition, a
  candidate whose state carries the same blocker will reproduce the same failure - prefer
  one whose state does not.
2.  Among candidates that escape the blocker, prefer the latest:  it retains the most sound
  work.  Choose an earlier one only when the trajectory shows the mistake was committed
  before it.
3.  Each candidate reports prior_rollbacks:  how many times that exact checkpoint was
  already restored in this task without solving it.  Re-restoring one repeats an
  intervention already known not to work; prefer a lower count when the counts differ.
4.  Return exactly one member of candidate_checkpoints; never invent or round an index.
</rules>
```

Figure 7: The two rollback target selection prompts. Stage 1 proposes the interval, Stage 2 chooses within it.

to a committed state or earlier action choice, the reviewer produces a rollback diagnosis f_t that is subsequently used by the restore-point selector.

Review urgency. The recovery verdict and the review schedule are intentionally separated. A branch may still be worth continuing while exhibiting uncertainty, repeated friction, or weak progress. Such cases can receive CONTINUE together with high urgency, causing the controller to revisit the branch after a shorter horizon. Productive trajectories instead receive lower urgency and are allowed to proceed for longer without interruption. Thus, the reviewer need not make a premature binary recovery decision merely to ensure that a risky branch is examined again soon.

Finally, agent-triggered and scheduled reviews share the same semantic reviewer but arise from complementary signals. An explicit rollback request exposes failure recognized by the executor itself, whereas scheduled review can identify stagnation or causal inconsistency that the acting agent has not recognized. This division underlies the hybrid review mechanism used in RIR: the agent can request immediate inspection when necessary, while adaptive scheduling provides an independent opportunity to detect failures that would otherwise persist.

A.3 ROLLBACK TARGET SELECTION

Rollback target selection proceeds in two stages. Both selectors are prompted with the task objective, the failure diagnosis produced by the review that triggered recovery, the executed trajectory, and the

Reflection memory prompt

```
<role>
You write the task memory after an attempt was rolled back:  why it failed, which beliefs
it invalidated, and which subgoals it genuinely attained.  What you write is everything the
next attempt will know about this one.
</role>

<rules>
Write in this order of care; apply every rule.
1.  Reasoning is a proposal, not an event.  Only an action payload with its returned
  observation establishes what was attempted, observed, or attained; do not turn subjects
  mentioned only in reasoning into facts or attained subgoals.
2.  Condition every cause.  Each failure cause names the condition under which the thing
  failed - the acting state, position, or missing prerequisite in effect.  Read
  restored_executor_state:  it reports the state the next attempt will actually start from,
  and when a repeated action had no effect, that state is the prime suspect.  An action
  that failed under one condition remains available under others, and the next attempt
  obeys your words verbatim:  a bare ``X does not work'' bans X everywhere.
3.  Hard ban on global verdicts.  No item may assert an unsupported global existence or
  impossibility claim (``X does not exist anywhere'', ``the task cannot be completed'',
  ``the search was exhaustive'', ``all candidates were checked''):  a failed search proves
  only that the examined candidates lacked the target, and candidates with no entry always
  remain.
4.  Invalidated beliefs take only specifics.  Only beliefs disproven by contradictory
  evidence, a verified error, or a task-contract violation qualify - never a behaviour
  category the task requires (such as exploration), never a claim that is merely unproven.
  For source or access failures, invalidate only the failed path and keep the candidate
  unverified.
5.  Attained subgoals are reusable history.  Record only intermediate results the abandoned
  attempt demonstrably reached, each with the means that reached it, so the next attempt
  can reuse the route.  State the outcome, not the motion:  an attempt that merely moved,
  looked, or failed attained nothing and gets no entry.  Most abandoned attempts attain
  nothing - returning an empty list is the normal case, not a failure to comply.
6.  Never claim the present.  History says what was once reached, never what holds now:
  the rollback undid the state, not the history.  Do not write a subgoal that would read as
  a description of the current situation.
7.  Replace, do not accumulate.  The failure analysis is a complete replacement of the
  previous one:  keep still-valid items, drop obsolete or contradicted ones, and never
  carry two conflicting explanations.  Attempt history is the opposite - append-only;
  earlier attempts' records are not yours to rewrite.
8.  Stay inside your boundary.  You write causes, invalidated beliefs and attained subgoals
  - never instructions (the next move is the generator's decision), never world facts or
  behavioural rules (they belong to the environment model, which the reviewer owns and
  which survives your patch).
</rules>
```

Figure 8: The prompt of Reflection Memory extraction.

Reflection Memory as read-only context. Figure 7 reproduces the role and rule sections of both selector prompts.

In the first stage, the coarse selector proposes an inclusive interval of candidate checkpoints. The interval is chosen to cover the portion of the trajectory most relevant to the diagnosed failure: its upper boundary should precede the decision that committed the agent to the failing course, while its lower boundary should avoid rewinding beyond the point at which additional rollback would provide materially different decision context. The selector is given the admissible checkpoint indices and, when applicable, a maximum interval width.

In the second stage, the fine selector chooses a single restore point from the admissible checkpoints within the localized interval after candidates known to correspond to dead ends have been excluded. In addition to the shared context, it receives the interval selected by the coarse stage together with the corresponding rationale, a compact description of the execution state restored by each candidate checkpoint, and the number of times each checkpoint has previously been restored during the current task.

A.4 Reflection Memory Extraction

Reflection Memory is updated once per rollback, after the restore point has been selected and before checkpoint restoration is executed. The reflection patcher receives the abandoned suffix, the failure

diagnosis produced by the review that triggered recovery, the current Reflection Memory, the index of the abandoned attempt, and the executor state stored at the selected checkpoint—that is, the state from which the next attempt will resume.

From this context, it extracts a concise summary of the abandoned attempt, the conditions that caused its failure, the hypotheses invalidated by the attempt, and the subgoals or milestones it successfully reached. These outputs are used to update the Attempt History and Failure Analysis fields of the Reflection Memory. The Task Objective remains fixed, while the Environment Model is not modified by the reflection patcher: environment knowledge is committed separately by the reviewer and therefore passes through this update unchanged. Figure 8 reproduces the role and rule sections of the LLM-based reflection patcher prompt.

B Proofs for the Recovery-Space Analysis

This section provides the definitions and proofs underlying Proposition 1 and Corollary 1 in Section 4.5. The comparison holds memory-update options fixed at each rollback depth and varies only the admissible depths.

Recovery configurations. At a recovery event on branch step t, let $\mathcal{I}_t$ be the admissible checkpoint indices and define

$$\mathcal{K}_t = \{\, t - r \mid r \in \mathcal{I}_t \,\}. \tag{14}$$

For $r = t - k$, let $\mathbb{M}(\mathcal{M}, \tau_{r:t})$ be the admissible memory configurations associated with that restore point. This family includes the unchanged memory $\mathcal{M}$ and the full update $\phi(\mathcal{M}, \tau_{r:t})$ from Equation 10. The proof does not require arbitrary combinations of partially updated fields; any additional configurations must be admitted equally by both policy classes.

The recovery operator is

$$\mathcal{R}_{k,\mathcal{M}^+} : (s_t, h_t, \mathcal{M}) \longmapsto (s_{t-k}, h_{t-k}, \mathcal{M}^+), \qquad (k, \mathcal{M}^+) \in \mathcal{C}_t, \tag{15}$$

where

$$\mathcal{C}_t = \left\{ (k, \mathcal{M}^+) \,\middle|\, k \in \mathcal{K}_t,\ \mathcal{M}^+ \in \mathbb{M}(\mathcal{M}, \tau_{t-k:t}) \right\}. \tag{16}$$

The corresponding operator family is

$$\mathfrak{R}_{\mathrm{RIR}}(t) = \left\{ \mathcal{R}_{k,\mathcal{M}^+} \,\middle|\, (k, \mathcal{M}^+) \in \mathcal{C}_t \right\}. \tag{17}$$

The dependence of the memory family on k is explicit: different restore points induce different abandoned suffixes.

Depth-restricted mechanisms. A restricted mechanism j permits depths in $\mathcal{K}_{j,t} \subseteq \mathcal{K}_t$ and therefore has configuration set

$$S_{j,t} = \left\{ (k, \mathcal{M}^+) \in \mathcal{C}_t \,\middle|\, k \in \mathcal{K}_{j,t} \right\}. \tag{18}$$

For every permitted depth, its memory options are identical to those of the unrestricted class. Examples include restart with $\mathcal{K}_{j,t} = \{t\}$, fixed-depth rollback with $\mathcal{K}_{j,t} = \{k_0\}$, and bounded-depth rollback with $\mathcal{K}_{j,t} = \{k \in \mathcal{K}_t : k \le b\}$, whenever these sets are admissible and nonempty. A restriction need not be strict at every event.

Policy classes and comparison conditions. Let Π_{RIR} denote the idealized class of history-dependent recovery policies with access to $\mathcal{C}_t$, and let Π_j denote the corresponding class restricted to $S_{j,t}$. Policies may randomize, but may use only information available at the current decision. The two classes otherwise have identical policy freedom.

Both classes share the initial task distribution, base agent, available information, review process, checkpoint-restoration semantics, non-recovery updates, and termination rules. They also share the same budgets and cost accounting: reproducing a configuration at the same history incurs the same cost and induces the same subsequent transition law. The restriction applies at every recovery history, with nonempty feasible configuration sets whenever a recovery decision is required.

Although indexed by the branch-local step t, the admissible sets may depend on the full recovery history and remaining budget. Budget counters are cumulative and are not reset when execution returns to an earlier checkpoint. The analysis concerns admissible policy behavior under these shared rules; it does not assign different costs merely because one policy class offers more choices.

Proof of Proposition 1. Take any $\pi_j \in \Pi_j$. At each recovery history, construct a policy $\widetilde{\pi}$ that assigns the same probabilities as π_j to configurations in $S_{j,t}$ and zero probability to configurations in $\mathcal{C}_t \setminus S_{j,t}$. Since $S_{j,t} \subseteq \mathcal{C}_t$, these decisions are admissible for the unrestricted class.

Starting from the shared initial distribution, the two policies induce the same distribution over subsequent histories. To see this, condition on any common history. At a recovery event, their configuration distributions agree; between recovery events, the base agent and other transition rules agree. Their conditional next-history distributions and incurred costs are therefore identical. Induction over chronological execution events establishes equality of their trajectory and budget-consumption laws. Consequently, every budget-feasible policy in Π_j is also available in Π_{RIR}, giving

$$\Pi_j \subseteq \Pi_{\mathrm{RIR}}. \tag{19}$$

This proves Proposition 1.

Proof of Corollary 1. For verifiable tasks, let $R_g \in \{0, 1\}$ indicate success and define $J(\pi) = \Pr_\pi(R_g = 1)$ under the shared task distribution and execution rules. For a nonempty policy class Π, let $V^\star(\Pi) = \sup_{\pi\in\Pi} J(\pi)$. By Proposition 1,

$$V^\star(\Pi_{\mathrm{RIR}}) = \sup_{\pi\in\Pi_{\mathrm{RIR}}} J(\pi) \geq \sup_{\pi\in\Pi_j} J(\pi) = V^\star(\Pi_j). \tag{20}$$

No maximizing policy need exist for this inequality to hold. The same argument applies to expected return whenever the objective is well-defined and bounded. This proves Corollary 1.

C Additional Experimental Details

C.1 Dataset and Metric Details

ALFWorld. ALFWorld (Shridhar et al., 2020) is a text-based embodied environment covering six task families: pick-and-place, pick-two, clean-and-place, heat-and-place, cool-and-place, and examine-under-light. We evaluate on test dataset from the official split. A task is considered successful only when the environment returns its success termination signal.

For an evaluation set containing N tasks, we report Success Rate (SR):

$$\mathrm{SR} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[z_i = 1], \tag{21}$$

where z_i indicates whether task i is successfully completed. Timeouts, execution errors, and unfinished episodes are counted as failures.

ScienceWorld. ScienceWorld (Wang et al., 2022) is a text-based scientific environment containing 30 task types with multiple environment variations. To avoid allowing task categories with many variations to dominate the aggregate results, we sample at most 10 test variations from each task type, resulting in 271 evaluation tasks. ScienceWorld assigns each trajectory a cumulative score $q_i \in [0, 100]$. A task is considered fully successful only when $q_i = 100$. In addition to SR, we report the average normalized Dense Reward (DR):

$$\mathrm{DR} = \frac{1}{N}\sum_{i=1}^{N} \frac{\max(0, q_i)}{100}. \tag{22}$$

DR complements binary success by capturing partial progress on trajectories that do not fully complete the task.

GAIA. GAIA (Mialon et al., 2024) evaluates general-purpose agents on open-domain tasks involving web search, code execution, file processing, and multi-tool reasoning. We evaluate on the dataset after filtering image- and audio-dependent instances from the public validation split and report results for overall performance.

For GAIA, an independent evaluator receives the task question x_i, reference answer $y_i^\star$, and agent prediction $\hat{y}_i$, and returns a binary correctness label:

$$z_i = J_{\mathrm{eval}}(x_i, y_i^\star, \hat{y}_i) \in \{0, 1\}. \tag{23}$$

GAIA Judge Prompt

```
You are an impartial evaluator. Judge whether the prediction satisfies the task given
the reference answer. Accept equivalent wording, formatting, units, and aliases where
appropriate. Ignore superficial differences that do not affect the correctness of the
answer.

Task: {question}
Reference answer: {reference_answer}
Prediction: {prediction}
Metadata: {task_metadata}

Return a structured judgment containing:

      • correct: whether the prediction satisfies the task;
      • score: confidence in the judgment, from 0 to 1;
      • reason: a brief rationale for the judgment.
```

Judge Response Schema

```
{
  "type": "object",
  "required": ["correct", "score", "reason"],
  "properties": {
    "correct": {
      "type": "boolean",
      "description":
        "Whether the prediction satisfies the task."
    },
    "score": {
      "type": "number",
      "minimum": 0.0,
      "maximum": 1.0,
      "description":
        "Confidence in the correctness judgment."
    },
    "reason": {
      "type": "string",
      "description":
        "Brief evaluation rationale."
    }
  }
}
```

Figure 9: Prompt used by the independent LLM-as-a-Judge for GAIA final-answer evaluation. The judge receives the task, reference answer, prediction, and task metadata, and does not observe the agent trajectory or participate in execution or recovery.

GAIA success rate is then computed over the complete evaluation subset using the same SR definition above. The evaluator has no access to the agent trajectory and does not participate in agent execution, review, restore localization, or Reflection Memory updates. Empty answers, malformed outputs, timeouts, and unrecoverable execution errors are counted as failures.

C.2 LLM-AS-A-JUDGE PROTOCOL

For GAIA, final-answer correctness is evaluated by an independent Qwen3.8-Max model. The evaluator is invoked only once after an episode terminates and is completely separated from the acting agent and all recovery components. Its input contains only the original task, the reference answer, the predicted answer, and minimal metadata required to interpret the expected answer format. The evaluator is instructed to ignore superficial differences in wording and judge whether the prediction is semantically consistent with the reference answer while satisfying all required answer constraints, as shown in Figure 9.

To prevent evaluation leakage, reference answers are never exposed to the acting agent, reviewer, selector, reflection memory patcher, or any baseline controller. Judge calls are used exclusively for evaluation and are therefore excluded from the inference cost of the compared methods.

Table 4: Model assignments used in our experiments. Recovery components always use the same backbone as the acting agent; the independent judge is used only for final GAIA evaluation.

Benchmark	Agent Backbone	Recovery Components	Judge
ALFWorld	Qwen3-14B / DeepSeek-V3 / Qwen3.8-27B	Same as backbone	Environment
ScienceWorld	Qwen3-14B / DeepSeek-V3 / Qwen3.8-27B	Same as backbone	Environment
GAIA	Qwen3-14B / DeepSeek-V3 / Qwen3.8-27B	Same as backbone	Qwen3.8-Max

C.3 Baseline Implementation Details

All baselines use the same backbone model, environment interface, task prompt, and tool set as RIR within each experimental condition. Differences are restricted to the recovery, reflection, or search mechanism required by each method. Whenever possible, we follow the official implementation and recommended configuration of the corresponding method. Components such as critics or evaluators use the same backbone as the acting agent unless otherwise stated.

ReAct. ReAct (Yao et al., 2022) serves as the base agent without an explicit recovery mechanism. The agent alternates between reasoning and environment actions using the current task context, without additional reflection memory or state rollback.

Self-Refine. Self-Refine (Madaan et al., 2023) performs information-level correction without restoring the environment state. We periodically invoke a critic over the current trajectory and inject its feedback into subsequent agent context. A refinement is triggered every 6 steps, with at most 8 refinement rounds per task. Once the critic determines that no correction is required, no additional refinement is performed until the next scheduled review.

Reflexion. Reflexion (Shinn et al., 2023) is implemented using repeated trials with cross-trial verbal reflection. Each task receives at most 2 trials. After a failed trial, the environment is reset to its initial state and the model summarizes the failed trajectory into a textual reflection that is provided to the next trial. Agent calls from all trials contribute to the same cumulative interaction budget.

LATS. LATS (Zhou et al., 2023) explores and evaluates multiple candidate trajectories through tree search. We use an expansion width of 3, at most 8 search iterations, and a simulation depth of 5. All model calls generated by branch expansion, evaluation, and simulation are included in the cumulative interaction cost.

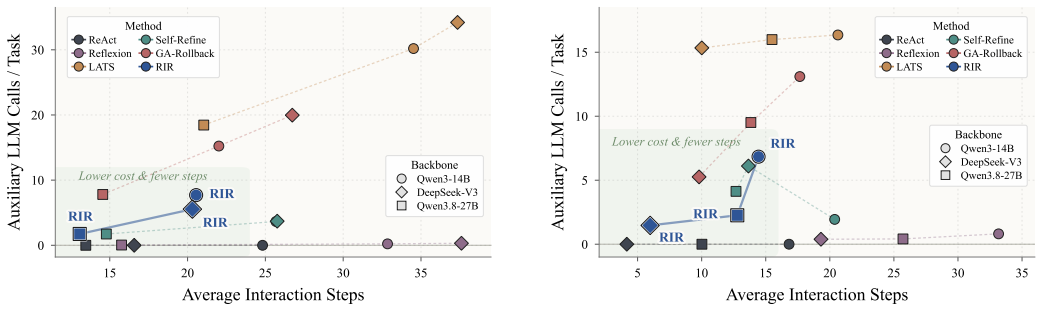
GA-Rollback. GA-Rollback (Li et al., 2025) periodically inspects the generated trajectory and restores execution to an earlier state when an unreliable step is detected. Following the published protocol, we use $\text{wait-}k = 6$ and a maximum rollback span of 4 per task.

RIR. The full RIR configuration enables Hybrid Adaptive Review, Coarse-to-Fine Restore Localization, and Rollback-Consistent Reflection Memory. For each backbone condition, the acting agent, reviewer, selector, and reflection memory patcher all use the same LLM. The initial review horizon is 6, with an adaptive range of $[4, 8]$.

C.4 Implementation and Fairness Details

Model configuration. We evaluate each method using Qwen3-14B, DeepSeek-V3, and Qwen3.8-27B as the backbone models. Within a given backbone condition, the acting agent and all method-specific auxiliary components use the same model. In particular, RIR does not rely on a stronger model for review, restore localization, or reflection extraction. GAIA final-answer evaluation uses the independent Qwen3.8-Max judge described in Appendix C.2.

State restoration. For ALFWorld and ScienceWorld, checkpoints are restored by resetting the environment and replaying the retained action prefix when a native persistent snapshot is not available. GAIA requires additional handling because tool calls may create or modify local files. A checkpoint therefore includes both the agent context and the task workspace state. When restoring a checkpoint, files created exclusively by the abandoned suffix are removed and modified files are restored to their checkpointed versions, preventing filesystem side effects from leaking across branches.



(a) ALFWorld (b) GAIA

Figure 10: Additional efficiency comparisons on ALFWorld and GAIA across multiple backbones. The horizontal axis denotes average interaction steps and the vertical axis denotes auxiliary LLM calls per task. Colors represent methods and marker shapes represent LLM backbones. Lower-left corresponds to lower interaction and auxiliary inference overhead. For Reflexion, interaction steps are summed across up to two trials. On GAIA, the 24-step limit applies separately to each trial, so the cumulative number of steps per task can exceed 24.

Table 5: Average number of state rollbacks per task on ALFWorld and GAIA.

Backbone	Method	ALFWorld	GAIA
Qwen3-14B	GA-Rollback	0.490	1.290
	RIR	**0.410**	**0.546**
DeepSeek-V3	GA-Rollback	**0.224**	0.787
	RIR	0.657	**0.184**
Qwen3.8-27B	GA-Rollback	0.049	0.880
	RIR	**0.036**	**0.408**

GAIA tools. All methods receive the same set of web search, web browsing, code execution, and file I/O tools. Tool outputs, attachment paths, and attachment-size limits are held fixed across methods. Neither prompts nor tool interfaces expose the reference answer during agent execution.

Execution budgets. We use benchmark-specific interaction limits to account for differences in task horizon and environment complexity. On ALFWorld, we set the maximum number of interaction steps to 50 and allow at most 20 rollbacks per task. On ScienceWorld, the maximum number of interaction steps is 100, with the rollback budget likewise set to 20. On GAIA, we use a maximum of 24 interaction steps and allow at most 20 rollbacks per task. Once either the interaction-step budget or the rollback budget is exhausted, no further interaction or rollback is permitted. All reported experimental results are averaged over three independent runs.

D Additional Results

D.1 Complete Efficiency Results

Figure 10 provides the corresponding efficiency comparisons on ALFWorld and GAIA. As in the ScienceWorld results in Figure 2, the horizontal axis denotes the average number of interaction steps and the vertical axis denotes auxiliary LLM calls per task. Colors distinguish methods, while marker shapes distinguish LLM backbones. Points closer to the lower-left indicate lower test-time overhead.

Table 5 reports the corresponding rollback frequency. Despite allowing up to 20 rollbacks per task, RIR invokes recovery infrequently in both benchmarks. It uses fewer rollbacks than GA-Rollback in five of the six backbone–benchmark settings; the only exception is ALFWorld with DeepSeek-V3, where both methods already exhibit low rollback frequency.

Table 6: Ablation results on ScienceWorld with Qwen3-14B. **SR** denotes success rate and **DR** denotes normalized dense reward. **Avg. Steps**, **Rev.**, **RB**, and **Aux.** report per-task averages of environment interaction steps, reviewer invocations, executed rollbacks, and auxiliary LLM calls, respectively. Executed rollbacks restore both the environment state and branch-local trajectory to an earlier checkpoint.

Variant	SR↑	DR↑	Avg. Steps	Rev.	RB	Aux.
Review-trigger variants						
Agent-only	34.32	0.435	53.4	5.39	0.69	7.70
Sched.-only (fix.)	33.58	0.491	45.8	7.10	0.54	9.01
Sched.-only (adap.)	34.32	0.468	46.9	8.48	0.75	11.27
Hybrid (fixed)	36.90	0.445	50.2	8.20	0.80	10.96
Restore-localization variants						
No rollback	33.95	0.441	43.1	8.30	/	11.44
Restart	35.79	0.454	54.4	10.56	1.17	14.07
Direct selection	36.16	0.490	49.4	9.27	0.86	12.80
Recovery-component variants						
Reflection only	33.95	0.441	43.1	8.30	/	11.44
Rollback only	35.42	0.439	48.7	9.47	1.04	13.21
RIR	36.67	0.448	51.5	9.90	0.98	13.33

Table 7: Ablation results on ScienceWorld with Qwen3.8-27B. **SR** denotes success rate and **DR** denotes normalized dense reward. **Avg. Steps**, **Rev.**, **RB**, and **Aux.** report per-task averages of environment interaction steps, reviewer invocations, executed rollbacks, and auxiliary LLM calls, respectively. Executed rollbacks restore both the environment state and branch-local trajectory to an earlier checkpoint.

Variant	SR↑	DR↑	Avg. Steps	Rev.	RB	Aux.
Review-trigger variants						
Agent-only	73.43	0.837	37.1	3.31	0.73	5.46
Sched.-only (fix.)	73.80	0.844	36.2	5.65	0.97	8.59
Sched.-only (adap.)	70.11	0.810	46.0	9.92	1.61	12.33
Hybrid (fixed)	76.38	0.862	38.8	7.08	1.15	8.63
Restore-localization variants						
No rollback	75.64	0.860	31.7	7.61	/	9.74
Restart	71.59	0.831	49.1	8.15	1.54	12.47
Direct selection	71.96	0.829	38.6	8.03	1.26	9.80
Recovery-component variants						
Reflection only	75.64	0.860	31.7	7.61	/	9.74
Rollback only	74.17	0.852	41.8	7.28	1.89	11.51
RIR	79.70	0.872	39.2	7.11	1.27	10.70

D.2 Additional Ablation Results

Across the three backbones, RIR maintains a consistent advantage over the non-RIR baselines in the main results (Table 1). To examine whether this advantage also holds at the level of individual design choices, we report component-level ablations on ScienceWorld for Qwen3-14B (Table 6) and Qwen3.8-27B (Table 7), complementing the DeepSeek-V3 ablations in the main text. The picture is consistent across all three backbones: the default RIR configuration—hybrid adaptive review, coarse-to-fine localization, and rollback-consistent reflection—attains the best or a closely comparable SR in every ablation group, and its margin over the constrained variants widens as the backbone becomes more capable. Because RIR delegates the open-ended *when*, *where*, and *what* decisions to the reviewer, a stronger reviewer can better exploit the additional flexibility, whereas on the weaker Qwen3-14B the more autonomous and the more constrained variants converge to comparable performance.

When to review. Combining agent-initiated and scheduled review is beneficial on both backbones: the two hybrid variants outperform every single-trigger variant. On Qwen3.8-27B, the full adaptive

RIR is clearly best (79.70% SR, 0.872 DR), exceeding Hybrid (fixed) by 3.32 points in SR. On Qwen3-14B, the two hybrid variants are essentially tied—Hybrid (fixed) reaches 36.90% SR and RIR 36.67%—while RIR retains a marginally higher DR (0.448 vs. 0.445). Adaptive triggering asks the reviewer to judge whether the trajectory has left a recoverable path and whether immediate intervention is preferable to continued exploration; a stronger reviewer answers these questions reliably enough to turn the added flexibility into a clear gain, whereas on Qwen3-14B adaptive and fixed scheduling perform comparably.

Where to recover. Coarse-to-fine localization yields the best SR on both backbones. On Qwen3.8-27B, RIR reaches 79.70% SR and 0.872 DR, well above direct selection (71.96%) and no rollback (75.64%). On Qwen3-14B, RIR again attains the highest SR (36.67%), ahead of direct selection (36.16%), restart (35.79%), and no rollback (33.95%); the last confirms that reflection without state restoration cannot reliably recover failed trajectories. Direct selection obtains a slightly higher DR on Qwen3-14B (0.490 vs. 0.448) but at a lower SR, so the two-stage selector provides the better success–progress balance. Its advantage is larger on the stronger backbone, where region localization and checkpoint selection are both more reliable and therefore compound rather than conflict.

Rollback vs. Reflection. Coupling state restoration with reflection is preferred on both backbones and on both metrics. On Qwen3.8-27B, RIR reaches 79.70% SR and 0.872 DR, above reflection only (75.64% SR, 0.860 DR) and rollback only (74.17% SR, 0.852 DR). On Qwen3-14B, RIR leads with 36.67% SR and 0.448 DR, exceeding rollback only (35.42% SR, 0.439 DR) and reflection only (33.95% SR, 0.441 DR). Restoring the environment and trajectory removes the consequences of an earlier erroneous action, while rollback-consistent reflection preserves transferable knowledge across attempts; the two mechanisms are complementary rather than redundant, and their combination is the strongest recovery-component configuration regardless of backbone capability.

Taken together, these ablations reinforce rather than qualify the design of RIR. On all three backbones, the default configuration is the best or a closely comparable choice in every ablation group, and no constrained variant dominates it: where a single-axis variant achieves a marginally higher DR on Qwen3-14B, it does so at a clear cost in SR. The advantage grows with backbone capability, consistent with the view that adaptive review, coarse-to-fine localization, and rollback-consistent reflection give a more capable reviewer more useful room to act. RIR is therefore best viewed as a general recovery framework whose full configuration transfers across backbone scales, with the largest gains realized on stronger models.

E Case Studies

We complement the aggregate results with a qualitative analysis of five representative trajectories, illustrating both successful recovery and characteristic failure modes of RIR. The selected cases span ALFWorld, ScienceWorld, and GAIA and are generated with DeepSeek-V3. We include three successful trajectories in which rollback contributes to subsequent task completion (Section E.1) and two unsuccessful trajectories that expose limitations in restore localization and reflection (Section E.2).

For successful cases, we consider trajectories that complete the task with at least one rollback, since trajectories without recovery provide limited evidence about the rollback mechanism itself. For failure analysis, we focus on trajectories with repeated recovery attempts, which make the source of unsuccessful recovery easier to isolate. To expose the full recovery dynamics of these trajectories, we use an enlarged maximum-step budget for the qualitative analysis, allowing repeated rollback-and-retry cycles to unfold more completely; this diagnostic setting is used only for the case studies and does not affect the main experimental results. Quoted trajectory content, including selector reasoning, reflection outputs, and Reflection Memory entries, is reproduced directly from the corresponding execution logs. Each case is followed by an analysis of the recovery behavior it illustrates, and Section E.3 summarizes the common patterns across all five trajectories.

Trajectory indices are branch-local and therefore may repeat after rollback; each rollback transition explicitly indicates the checkpoint from which execution resumes.

E.1 GOOD CASES

E.1.1 G1 — ALFWORLD, COMMITTED STATE ERROR

G1 ALFWorld

Task. `put a soapbottle in toilet`

Initial exploration idx 0--5

0 **ACT** `go to cabinet 1`
OBS You arrive at cabinet 1. The cabinet 1 is closed.

1 **ACT** `open cabinet 1`
OBS You open the cabinet 1. The cabinet 1 is open. In it, you see a candle 2, and a cloth 2.

2 **ACT** `go to cabinet 2`
OBS You arrive at cabinet 2. The cabinet 2 is closed.

3 **ACT** `open cabinet 2`
OBS You open the cabinet 2. The cabinet 2 is open. In it, you see a candle 3, and a candle 1.

4 **ACT** `go to cabinet 3`
OBS You arrive at cabinet 3. The cabinet 3 is closed.

5 **ACT** `open cabinet 3`
OBS You open the cabinet 3. The cabinet 3 is open. In it, you see a spraybottle 1.

Abandoned branch idx 6--9

6 **ACT** `take spraybottle 1 from cabinet 3`
OBS You pick up the spraybottle 1 from the cabinet 3.

7 **ACT** `go to toilet 1`
OBS You arrive at toilet 1. On the toilet 1, you see a soapbar 2, and a toiletpaper 2.

8 **ACT** `move spraybottle 1 to toilet 1`
OBS *"You move the spraybottle 1 to the toilet 1."*

9 **ACT** [ROLLBACK requested]
WHY *"The task was to put a soapbottle in the toilet, but the spraybottle was moved instead. The soapbottle was not found in any of the cabinets checked so far."*

▶ The action at idx 8 has already taken effect in the environment—this is not a rejected or no-op action.

Recovery approved $C_9 \rightarrow C_5$

"The root cause of the failure was the decision to take the spraybottle from cabinet 3 at step 6. Restoring to step 5 (before opening cabinet 3) allows for a re-evaluation of the contents of cabinet 3 and avoids the mistake of taking the wrong item. This checkpoint is the earliest and latest that avoids the error, and it has no prior rollbacks."

Reflection Memory retained across rollback 4 entries

OBSERVED cabinet 1: candle 2, cloth 2 | cabinet 2: candle 3, candle 1 | cabinet 3: spraybottle 1

AFFORDANCE opening a cabinet: requires the cabinet to be closed

Recovered branch idx 5--9

5 **ACT** `open cabinet 3` (replayed)
OBS You open the cabinet 3. The cabinet 3 is open. In it, you see a spraybottle 1.

6 **ACT** `go to countertop 1` (unexplored)
OBS You arrive at countertop 1. On the countertop 1, you see a soapbottle 1, and a spraybottle 2.

7 **ACT** `take soapbottle 1 from countertop 1`
OBS You pick up the soapbottle 1 from the countertop 1.

8 **ACT** `go to toilet 1`
OBS You arrive at toilet 1. On the toilet 1, you see a soapbar 2, and a toiletpaper 2.

9 **ACT** [FINAL] `move soapbottle 1 to toilet 1`
OBS You move the soapbottle 1 to the toilet 1.

▶ Previously searched cabinets are not revisited. ✓ **success**

Figure 11: **G1.** RIR recovers from a committed state error in ALFWorld. Rollback restores the environment to the state before the incorrect object was taken, while previously acquired exploration knowledge remains available through Reflection Memory. The recovered branch then searches an unexplored location and completes the task.

Analysis. G1 provides a direct example of RIR's central asymmetry between execution state and persistent knowledge. The incorrect action at idx 8 has already modified the environment, so textual correction alone cannot undo its consequence. Rolling back to C_5 removes the committed spraybottle branch, while Reflection Memory preserves what was learned from the preceding exploration. After replaying the checkpointed cabinet action, the agent does not return to cabinets 1 or 2 and instead explores the previously unvisited countertop, where it finds the correct soapbottle. Recovery therefore avoids both failure modes of the two simpler alternatives: continuing from the corrupted state and restarting without retaining useful information from the failed attempt.

E.1.2 G2 — ScienceWorld, precondition restoration and self-correction

G2 ScienceWorld

Task. `Find a(n) non-living thing. First, focus on the thing. Then, move it to the orange box in the bedroom.`

Initial exploration idx 0--5

0 **ACT** `teleport to bedroom`
OBS You teleport to the bedroom.

1 **ACT** `look around`
OBS This room is called the bedroom. In it, you see: … a finger painting a orange box (containing nothing) a table. On the table is: nothing.

2 **ACT** `look at orange box`
OBS a orange box (containing nothing)

3 **ACT** `teleport to workshop`
OBS You teleport to the workshop.

4 **ACT** `look at table`
OBS a table. On the table is: a battery a blue wire a green wire a orange wire a red light bulb, which is off …

5 **ACT** `focus on battery`
OBS You focus on the battery.

Misdirected branch idx 6--18

6 **ACT** `move battery to orange box in bedroom`
OBS *"No known action matches that input."*

7 **ACT** `teleport to bedroom`
OBS You teleport to the bedroom.

8 **ACT** `move battery to orange box`
OBS *"No known action matches that input."*

9 **ACT** [ROLLBACK requested] denied
WHY *"The command 'move battery to orange box' is not recognized . . . Need to find the correct syntax or object to move."*

10 **ACT** `pick up battery`
OBS *"No known action matches that input."*

12 **ACT** `look at table`
OBS a table. On the table is: nothing (bedroom table—battery is in the workshop)

idx 11, 13 omitted: `look at battery` *(unrecognized), repeated* `look around`

14 **ACT** `focus on finger painting`
OBS You focus on the finger painting.

15 **ACT** `move finger painting to orange box`
OBS *"You move the finger painting to the orange box."* (succeeds environmentally)

idx 16–17: `think` $\times 2 \rightarrow$ *"No env action was executed."*

18 **ACT** [ROLLBACK requested]
WHY *". . . the task requires focusing on a non-living thing. The finger painting may not qualify, and the agent needs to revisit the workshop to find a valid non-living object."*

▶ Two independent failures are entangled here: an unmet spatial precondition (the battery is not co-located with the agent), and a self-identified semantic misjudgment (finger painting may not qualify as "non-living"). Note that the environment reports the former as *"No known action matches that input"*—a *syntax* error message— which is why both the agent's request at idx 9 and the retained memory attribute the failure to syntax rather than to location.

Recovery approved $C_{18} \rightarrow C_5$

"The root cause of the failure is the incorrect focus on the finger painting, which may not qualify as a non-living thing. Restoring to step 5 allows the agent to correctly identify the battery as a non-living thing . . . "

Reflection Memory retained across rollback 4 entries

OBSERVED battery: is a non-living thing

AFFORDANCE focusing on an object: requires the object to be present and visible
moving an object: requires correct syntax or method to execute
moving an object: requires correct syntax or method to execute, as demonstrated by the failed attempts to move the battery (near-duplicate)

Recovered branch idx 5--8

5 **ACT** `focus on battery` (replayed)
OBS You focus on the battery.

6 **ACT** `pick up battery` (same command as idx 10)
OBS *"You move the battery to the inventory."*

7 **ACT** `teleport to bedroom`
OBS You teleport to the bedroom.

8 **ACT** [FINAL] `move battery to orange box`
OBS You move the battery to the orange box.

▶ The identical `pick up battery` command failed at idx 10 and succeeded at idx 6 after the rollback—the only change is spatial co-location, not knowledge. The rollback therefore repaired a precondition that neither the agent's request nor the selector's rationale had explicitly identified. ✓ **success**

Figure 12: **G2.** RIR restores an unmet spatial precondition in ScienceWorld. The same `pick up battery` command that fails while the agent is in the bedroom succeeds after rollback restores the

agent to the workshop. The trajectory also contains both a denied and an approved rollback request, showing that agent-initiated recovery remains subject to reviewer adjudication.

Analysis. G2 shows that state restoration can recover a latent execution precondition even when the failure diagnosis is incomplete. At idx 10, `pick up battery` fails after the agent has moved from the workshop to the bedroom; after rollback, the identical command succeeds from the restored workshop state. The environment reports the original failure as *"No known action matches that input,"* causing both the agent and Reflection Memory to interpret it primarily as a syntax problem. Nevertheless, restore localization returns execution to a state in which the missing spatial precondition is satisfied. The case also illustrates Hybrid Adaptive Review: the first agent-issued rollback request is rejected, whereas the later request is approved after the branch develops into a clearer semantic misjudgment. Thus, RIR can benefit from state-level recovery even when the textual diagnosis does not fully identify the underlying execution condition.

E.1.3 G3 — GAIA, HORIZON-TRIGGERED ABANDONMENT OF A DEAD TECHNICAL PATH

G3 GAIA

Task. *Which of the text elements under CATEGORIES in the XML would contain the one food in the spreadsheet that does not appear a second time under a different name?*

`$ZIP` denotes the task's attachment archive; long paths are abbreviated.

Exhausted technical path `idx 0--5, all ACT`

0 **ACT** `unzip -l "$ZIP"`
OBS `food_duplicates.xls` (28160 B), `CATEGORIES.xml` (23933 B)

1 **ACT** `unzip -p "$ZIP" "food_duplicates.xls" > /tmp/f.xls && file /tmp/f.xls`
OBS Composite Document File V2 Document, … Creating Application: Microsoft Excel (legacy `.xls`)

2 **ACT** `pip install xlsx2csv && xlsx2csv /tmp/f.xls /tmp/f.csv`
OBS *"/bin/sh: pip: command not found"*

3 **ACT** `python3 -c "import pandas as pd; df = pd.read_excel('/tmp/f.xls'); print(df.to_string())"`
OBS *"ModuleNotFoundError: No module named 'xlrd' … Missing optional dependency 'xlrd'."*

4 **ACT** `file /tmp/f.xls` (identical to idx 1)
OBS Composite Document File V2 Document, … (byte-identical output)

5 **ACT** `ssconvert /tmp/f.xls /tmp/f.csv`
OBS *"/bin/sh: ssconvert: command not found"*

▶ No step in this trace has `action_type=ROLLBACK`: the agent never requests recovery. The re-execution at idx 4 of the command already issued at idx 1, returning byte-identical output, is what the reviewer registers as the attempt having stopped learning.

Horizon-triggered recovery $C_6 \rightarrow C_2$

REVIEWER *"The attempt has stopped learning; repeated actions with no new observations or progress. The environment lacks the necessary tools (pip, ssconvert) to proceed …"* (`HORIZON_REACHED`, urgency 0.8)

SELECTOR *"The earliest checkpoint (index 2) where the issue was first encountered (pip not found) is the latest point that still escapes the failure."* (range [2, 2], single candidate)

Reflection Memory retained across rollback `1 entry`

AFFORDANCE Bash command execution: Requires the command to be available in the environment (e.g., pip, ssconvert).

INVALIDATED *"The spreadsheet can be directly read or converted using available tools in the current environment."*

Recovered branch idx 2--4

2 **ACT** `unzip -p "$ZIP" "CATEGORIES.xml"`
OBS `<w:wordDocument xmlns:aml=... xmlns:wpc=...` (WordML boilerplate; the CATEGORIES text elements are not reached)

3 **ACT** `unzip -p "$ZIP" "food_duplicates.xls" > /tmp/f.xls && strings /tmp/f.xls | grep -v "\x00"`
OBS … chickpea garbanzo bean aubergine eggplant zucchini courgette … congee rice porridge **turtle soup** granola tofu soy curds …

4 **ACT** [FINAL] *"… "Soups and Stews", which includes "turtle soup"."*
OBS Final answer produced.

▶ The recovered branch does not repair the parsing toolchain—it abandons it. `strings` extracts the paired food names directly from the binary spreadsheet, leaving `turtle soup` as the unpaired entry. ✓ **success**

Note on necessity

Unlike G1/G2, the failed commands here did not corrupt any environment state—they simply failed. An agent that switched strategy *without* rolling back could plausibly have succeeded too. Rollback's benefit in this case is discarding a context saturated with four dead-end attempts, not undoing a side effect.

Figure 13: **G3.** A GAIA recovery triggered by the adaptive review horizon rather than by an agent-issued rollback request. After several unavailable parsing tools lead to the same technical dead end, recovery returns execution to an earlier checkpoint and the next branch adopts a different, dependency-free strategy.

Analysis. G3 illustrates the complementary role of scheduled review when failure is not recognized by the acting agent itself. No `ROLLBACK` action is issued during the failed branch: the agent continues trying increasingly redundant parsing routes, including re-running a command that produces an identical observation. The horizon-triggered reviewer detects this loss of information gain and abandons the tool-dependent parsing strategy. After recovery, the agent switches to dependency-free shell operations and extracts the required evidence without repairing the unavailable parsing toolchain. Unlike G1 and G2, rollback does not undo a persistent environmental side effect in this case; its primary benefit is to terminate an unproductive branch and reconstruct the decision context around a different strategy. G3 therefore mainly demonstrates the value of RIR's external review channel rather than the strict necessity of physical state restoration.

E.2 Bad Cases

E.2.1 B1 — ALFWorld, incorrect reflection persists across attempts

B1 ALFWorld Failure

Task. `put a clean cloth in countertop.`

Attempt 1 — the failure condition is created `idx 0--17`

0 **ACT** `go to handtowelholder 1`
OBS You arrive at handtowelholder 1. On the handtowelholder 1, you see a handtowel 1.

1 **ACT** `take handtowel 1 from handtowelholder 1`
OBS You pick up the handtowel 1 from the handtowelholder 1. (the single hand slot is now occupied)

2 **ACT** `go to sinkbasin 1`
OBS You arrive at sinkbasin 1. On the sinkbasin 1, you see a soapbar 1.

3 **ACT** `clean handtowel 1 with sinkbasin 1`
OBS *"Nothing happens."*

4 **ACT** [ROLLBACK requested] **denied**

6 **ACT** `take soapbar 1 from sinkbasin 1` (hand still holds the handtowel)
OBS *"Nothing happens."*

10 **ACT** `go to countertop 1`
OBS On the countertop 1, you see a candle 1, **a cloth 3**, and a spraybottle 1. (the task target is present)

11 **ACT** `take cloth 3 from countertop 1`
OBS *"Nothing happens."*

idx 5, 7–9, 12–16 omitted: further `take`/`examine` attempts on towel 1, spraybottle 1, cloth 3 — all "Nothing happens."

17 **ACT** [ROLLBACK requested] (approved: rollback #1)
WHY *"After multiple attempts, I have not found a way to clean t. . . "*

▶ The critical state change occurs at idx 1: the hand becomes occupied before any subsequent interaction is attempted. Every later `take` in this attempt returns the same undifferentiated signal *"Nothing happens."*, which the environment also emits for inapplicable actions—the agent cannot distinguish a full hand from a non-interactable object.

First diagnosis `lesson emitted at idx 17`

"Cleaning the handtowel with the sinkbasin failed because the sinkbasin lacked the required cleaning capability (no observable cleaning effect). Taking the soapbar from the sinkbasin failed, possibly due to unmet preconditions (e.g., the soapbar was not interactable)."

▶ The review reaches the correct abstraction—*unmet preconditions*— but instantiates it on the sinkbasin and the target objects rather than on the agent's occupied hand. The same run contains 80 successful `clean ... with sinkbasin` actions across all controllers, so the sinkbasin does possess the capability the review denies it.

Early recovery cycles `rollback #1:` $C_{17} \to C_2$ `rollback #2:` $C_{13} \to C_2$

#1 *"Restoring to step 2 (going to sinkbasin 1) avoids the failed cleaning action and allows reassessment of the cleaning approach without repeating the failure."*

#2 *"Checkpoint 2 is the latest checkpoint that avoids the failed cleaning action with the sinkbasin (step 3) and **retains the handtowel** for alternative cleaning attempts."*

▶ Both rollbacks remove the failed cleaning action, but restore a checkpoint at which the agent is still holding the handtowel. Rollback #2 makes this explicit: retaining the handtowel is treated as preserving sound work. The blocking condition therefore survives the early recoveries.

Attempt 4 — rollback finally precedes the true cause `rollback #4:` $C_{13} \to C_0$, `range` $[0, 1]$

*"Restoring to checkpoint 0 **(before taking the handtowel)** allows exploring alternative methods to clean the cloth, as checkpoint 1 leads directly to the failure due to the lack of viable cleaning tools."*

▶ This selector rationale correctly identifies acquisition of the handtowel as the action that must be undone and restores a state in which the hand is empty. It also contrasts with rollback #2, which preserved the same action as sound work.

Attempt 4 — the failure nevertheless reappears `after restoring` C_0

0 **ACT** `go to handtowelholder 1`
OBS On the handtowelholder 1, you see a handtowel 1.

1 **ACT** `take handtowel 1 from handtowelholder 1` (the blocking action, re-issued verbatim)
OBS You pick up the handtowel 1 from the handtowelholder 1.

2 **ACT** `go to sinkbasin 1`
OBS On the sinkbasin 1, you see a soapbar 1.

3 **ACT** `clean handtowel 1 with sinkbasin 1`
OBS *"Nothing happens."*

4 **ACT** [ROLLBACK requested] (approved: rollback #5, $C_4 \rightarrow C_2$)

▶ The physical rollback is now deep enough: the agent begins again with an empty hand and the original dead end is no longer entailed by the restored environment state. It nevertheless re-issues the blocking action immediately, while the persistent memory continues to discourage `cloth 3`.

Collapse into a two-step limit cycle `rollbacks #5 and #6, both` $C_4 \rightarrow C_2$`, depth 2`

2 **ACT** `go to sinkbasin 1`
3 **ACT** `clean handtowel 1 with sinkbasin 1` → *"Nothing happens."*
4 **WHY** *"Cleaning the handtowel with the sinkbasin failed,* ***as established in earlier attempts****..."*
↪ restored to C_2; the identical two steps are issued again

▶ The agent states that the action is already known to fail and issues it regardless. Restored to C_2, it stands at the sinkbasin holding the handtowel, while its retained knowledge provides no productive alternative.

Reflection Memory accumulated across failed attempts `5 of 7 entries`

3 taking an item: requires the item to be interactable (e.g., **the soapbar 1 and cloth 3** are not interactable)
4 taking an item: requires the item to be interactable (e.g., **the spraybottle 1 and cloth 3** are not interactable)
5 taking an item: requires the item to be interactable (e.g., **the cloth 3 and spraybottle 1** are not interactable) (same content as #4, order reversed)
6 taking an item: requires the item to be interactable (e.g., **the cloth 3** is not interactable)
7 taking an item: requires the item to be interactable (e.g., **the towel 1** is not interactable) (logged with `restated=1`)

▶ `cloth 3`—the correct task target—is recorded as non-interactable in four of these five entries. Entries #4 and #5 carry identical information and differ only in the order in which the two objects are listed; the deduplication key is literal, so both are admitted. Entry #7 is flagged as a restatement and retained anyway. Persistent memory therefore increasingly reinforces the incorrect interpretation formed during the failed attempts.

Restore-point repetition across the episode `6 of 9 rollbacks return to` C_2

#	range	→	selector's stated reason
2	[1, 2]	2	*"lower `prior_rollbacks` count* ***(1)*** *compared to checkpoint 1* ***(0)****"*
3	[2, 4]	2	*"lowest `prior_rollbacks` count* ***(2)*** *compared to other candidates"*
7	[1, 2]	2	*"lower `prior_rollbacks` count compared to checkpoint 1"*

▶ The field is intended to discourage repeatedly restoring the same checkpoint. In these examples, however, the selector treats the larger prior-rollback count as preferable, contributing to repeated restoration of C_2.

Outcome

Budget exhausted at 100 steps. Across the episode: 8 executions of `clean handtowel 1 with sinkbasin 1`, 5 of `take cloth 3 from countertop 1`, all returning *"Nothing happens."*; checkpoint revisit rate 44.44%, no-op patch rate 36.36%, restated-knowledge rate 41.67%. **✗ failure**

Figure 14: **B1.** An early diagnostic error propagates across successive recovery attempts. Initial rollbacks preserve the occupied-hand state responsible for the failure, while repeated failed interactions introduce misleading claims about the valid target into Reflection Memory. Even when

rollback #4 finally restores a state before the original blocking action, the retained knowledge helps reconstruct the same failed branch, after which execution collapses into a short recovery cycle.

Analysis. B1 exposes a compounding failure across both restore localization and persistent reflection. The initial observation, *"Nothing happens,"* is ambiguous, and the reviewer incorrectly attributes the failure to the sinkbasin or target objects rather than to the agent's occupied hand. Early rollbacks therefore preserve the handtowel as valid progress, while repeated failed interactions are progressively generalized into Reflection Memory entries that mark `cloth 3` as non-interactable. Restore localization also becomes increasingly repetitive, with six of nine rollbacks returning to C_2.

Rollback #4 isolates the role of memory particularly clearly. It correctly restores C_0, before the handtowel is acquired, and therefore removes the original blocking state. Yet the agent immediately reconstructs the same branch while the retained Reflection Memory discourages the valid cloth target. B1 therefore shows that successful rollback requires more than reaching a causally valid checkpoint: the knowledge carried across the rollback boundary must also remain consistent with the alternatives available in the restored state.

E.2.2 B2 — ScienceWorld, recovery without a concrete repair

B2 ScienceWorld Failure

Task. `Use chemistry to create 'red paper'. A recipe and some ingredients might be found near the workshop. When done, focus on the red paper.`

Attempt 1 — the recipe is obtained and read `idx 0--5`

0 **ACT** `teleport to workshop`
OBS You teleport to the workshop.

1 **ACT** `look around`
OBS … you see: the agent a substance called air **A blank paper A recipe titled instructions to make red paper** a table. On the table is: a battery, … a ultra low temperature freezer.

2 **ACT** `pick up recipe` (short form, not the displayed name)
OBS *"You move the recipe to the inventory."* (succeeds)

3 **ACT** `read recipe in inventory`
OBS *"The recipe reads: To make red paper, you need to **mix** paper, red paint."*

4 **ACT** `look around` (repeated)

5 **ACT** `pick up blank paper`
OBS *"No known action matches that input."*

▶ Two facts are established in the first four steps and never revised. The recipe explicitly names the required operation, `mix`. Moreover, `pick up recipe` succeeds at idx 2 using a short form rather than the displayed name *"A recipe titled instructions to make red paper"*, showing that short-form reference can work in this environment.

Early reflection focuses on object reference `across recovery cycles`

"Interacting with 'A blank paper' failed due to unrecognized inputs while in the workshop."

Reflection Memory: *"picking up an item: requires the item to be present and accessible **with the correct name**"; "interacting with paper: requires the correct action name or method to interact."*

▶ The diagnosis remains at the level of an invalid name or method. It does not instantiate the concrete alternative already present in the recipe: `mix`.

Attempt 8 — the missing ingredient is acquired idx 15--23

15 **ACT** `teleport to art studio`
OBS You teleport to the art studio.

17 **ACT** `pick up wood cup containing red paint`
OBS *"You move the wood cup to the inventory."* (succeeds)

21 **ACT** `move wood cup containing red paint to table`
OBS *"You move the wood cup to the table."*

23 **ACT** `move A blank paper to wood cup containing red paint`
OBS *"No known action matches that input."*

▶ The red paint is located in the art studio rather than the workshop. The agent successfully acquires it and returns it to the relevant setting. At this point the recipe and required ingredient are available, but the synthesis operation itself remains untried.

Recovery cycles repeatedly return to the recipe `7 of 14 rollbacks restore idx 3`

restore target	idx 3 (×7) idx 2 (×2) idx 4 (×2) idx 0, 17, 18 (×1 each)
recurrent pattern	rollback → restore near the recipe → retry a paper reference → unrecognized → reflect on "correct name or method" → rollback

▶ Restoring to idx 3 is itself a defensible choice: it retains the recipe while discarding failed interactions. However, the information carried into the next attempt continues to prioritize the reference hypothesis rather than a new operation. Of the 150 executed steps, 68 are replays of previously executed actions.

Reference search dominates the episode `full-trajectory aggregate`

count	action	result
14	`look at blank paper`	*"No known action matches that input."*
10	`look at A blank paper`	*same*
12	`pick up blank paper`	*same*
3	`pick up A blank paper`	*same*

= 39 attempts spanning both verbs × both spellings of the name

0	`mix ...`	**never issued**

▶ The agent repeatedly varies the verb and spelling of the paper reference. The operation explicitly named in the recipe—which is read thirteen times over the episode—is never attempted.

Reflection Memory grows without adding an actionable hypothesis `9 entries, restated rate 66.67%`

1 picking up an item: requires the item to be present and accessible
2 picking up an item: requires the item to be present and accessible **in a specific state or location**
3 picking up an item: requires the item to be present and accessible **with the correct name** (each entry extends the previous one)
8 workshop: holds exactly: a blank paper, a table with ... a switch (off), **a yellow light bulb (off), a wood cup (containing red paint)**
9 workshop: holds exactly: a blank paper, a table with ... a switch (off), **a wood cup (containing red paint), a yellow light bulb (off)** (identical content, last two items transposed)

▶ Entries 1–3 restate the same general hypothesis with increasing specificity, while entries 8 and 9 differ only in list order. The recorded restatement rate is 66.67%: the memory grows, but the set of actionable alternatives does not.

Outcome

Budget exhausted at 150 steps (`termination_reason = max_total_steps`). The recipe is read 13 times. The agent issues `look at`, `pick up`, `pour`, `use ... on`, and `move ... to` against the paper in multiple formulations—39 reference attempts in total— and **never issues `mix`**. No-op patch rate 64.29%, checkpoint revisit rate 35.71%. **✗ failure, reward 0.33** *(partial credit consistent with the recipe located and the ingredient acquired, synthesis not completed)*

Figure 15: **B2.** The agent obtains a recipe that explicitly specifies `mix` and later acquires the required red paint, yet subsequent recovery cycles continue to focus on how the paper should be ref-

erenced. Across the full episode, the agent tries 39 reference variants but never issues the operation provided by the recipe.

Analysis. B2 illustrates a different failure mode from B1: the system repeatedly diagnoses the problem at the wrong level of abstraction. After the first failed paper interaction, reflection focuses on finding the "correct name or method." Yet the trajectory already contains evidence against a pure naming explanation—`pick up recipe` succeeds at idx 2 despite using a shortened reference—and the recipe itself explicitly supplies the missing operation, `mix`. Later attempts also acquire the red paint, leaving the synthesis operation as the unresolved step. Nevertheless, recovery repeatedly returns to the same reference-search hypothesis, producing 39 naming variants and no `mix` action.

Unlike B1, the restored checkpoints themselves are generally reasonable. The failure lies primarily in the information carried forward: Reflection Memory repeatedly elaborates the same hypothesis without producing a concrete alternative that changes the next action. B2 therefore shows that useful reflection must be not only consistent with the trajectory, but actionable enough to alter the decision process after recovery.

E.3 Summary of the Case Studies

Across the successful trajectories, the three components of RIR play complementary roles. G1 demonstrates why explicit state restoration is valuable once an erroneous action has already changed the environment, while Reflection Memory prevents useful exploration from being discarded with that state. G2 shows that restoring an earlier checkpoint can recover an execution precondition that textual feedback alone does not repair. G3 complements these examples on the *when* dimension: scheduled review can terminate an unproductive branch even when the acting agent never requests recovery. Together, these cases illustrate the advantage of treating recovery as a joint decision over *when* to intervene, *where* to resume, and *what* information should survive.

The failed trajectories expose the corresponding requirements. B1 shows that incorrect failure attribution can propagate through both restore selection and persistent memory, remaining harmful even after execution is eventually restored to an appropriate state. B2 shows that a broadly plausible reflection may still be ineffective when it fails to produce an actionable alternative. Effective recovery therefore requires state restoration and information retention to reinforce one another: rollback must remove the condition that made the previous branch fail, while reflection must make the next attempt meaningfully different. This is the central advantage targeted by RIR over either mechanism in isolation: execution can return to a useful prior state without discarding reusable knowledge acquired along the abandoned branch.