# ERRSI: LOW-COST AGENT SELF-EVOLUTION VIA ELASTIC ERROR LOCALIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Agent recursive self-improvement (RSI) attracts growing interest as a way to improve agents through continued interaction. A crucial component of this process is accumulating and reusing experience to guide subsequent decisions. However, continually adding memories and instructions can increase recurring context costs and introduce irrelevant guidance across task environments. In this work, we propose ErRsi, a two-stage RSI framework with fixed model weights. In the first stage, to extract actionable evidence within context limits, we propose elastic error localization and organize diagnoses and update feedback into error lineages. In the second stage, to translate this evidence into behavioral improvement, we introduce targeted revision with conditional guidance and paired execution for update selection. On Who&When, localization achieves 30.43% exact-step accuracy at 9B and 36.96% at 27B, exceeding representative methods in the matched 9B evaluation. Diagnostic-input comparisons show that localized evidence supports effective revisions. On WebShop, complete-protocol token consumption falls by 44.67–69.77% relative to adapted Agentic Context Engineering (ACE). Frozen WebShop-to-ALFWorld transfer remains within one task of initial success across three model scales, with 75.39–78.64% fewer target-execution tokens than ACE. Together, these results demonstrate accurate error localization, efficient RSI, and low interference across benchmarks.

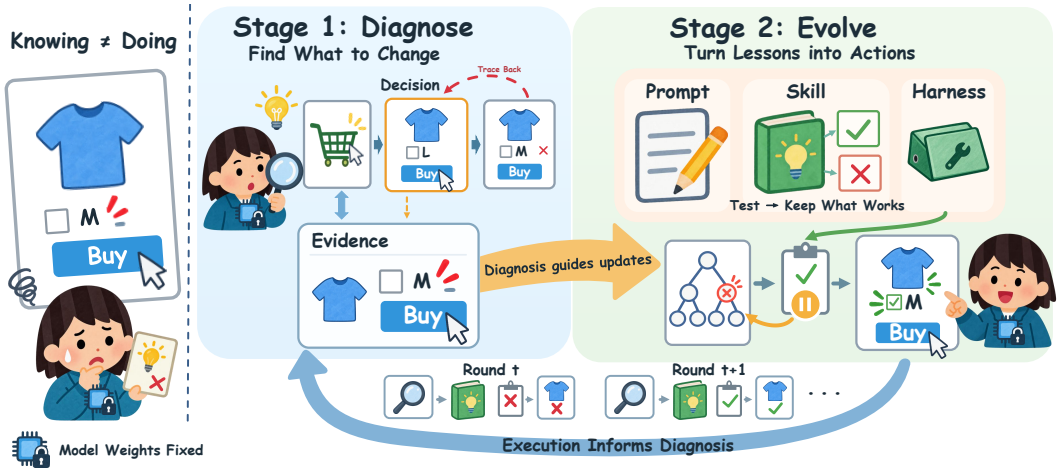


Figure 1: ErRsi learns from errors with fixed model weights. Stage 1 localizes failures. Stage 2 revises external behavior, evaluates changes, and feeds execution outcomes into the next round.

## 1 INTRODUCTION

Agent recursive self-improvement (RSI) attracts growing interest as a way to improve agents through continued interaction with their environments. As agents repeatedly perform tasks, their successes and failures provide experience that can inform subsequent decisions. Consequently, learning from that experience is crucial to agent self-evolution. However, execution histories do not automatically



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.

provide reusable guidance, creating a need to transform interactions into experience that agents can retain and apply.

To bridge this gap, recent approaches develop mechanisms for accumulating experience without updating model weights. *Reflexion* stores feedback from previous attempts, while *Voyager* accumulates executable skills for reuse (Shinn et al., 2023, Wang et al., 2024). *Dynamic Cheatsheet* and *Agentic Context Engineering (ACE)* further organize experience into evolving contextual guidance (Suzgun et al., 2026, Zhang et al., 2026b). However, continually adding experience raises two challenges when an agent operates across different task environments.

The first challenge is **deciding when accumulated experience should influence a decision**. The applicability of learned guidance depends on the task conditions under which it is acquired. Applying such experience beyond its scope can turn agents into narrow specialists that excel within one domain but perform poorly across the diverse tasks encountered in real-world deployment. For example, a lesson about selecting product options supports purchasing decisions, but carrying its details into every household-task decision consumes context without establishing relevance. Recent cross-domain memory studies confirm that experience transfer depends on its representation and match to the target task (Kim et al., 2026, Wang et al., 2026a). As a result, improving a source-benchmark score alone does not establish whether the learned experience remains useful when the task environment changes.

The second challenge is **the recurring cost of accumulated experience**. Across a continuing workload, hundreds or thousands of calls repeatedly process inherited guidance, multiplying even modest additions to each prompt. Many existing approaches pursue improvement by adding more instructions, memories, and reflections, allowing context costs to keep growing even when the marginal performance gains diminish. Learning also consumes context: long trajectories exceed practical analysis windows, and repeatedly presenting complete histories makes later revisions increasingly expensive. Efficient self-evolution therefore requires control over both learning evidence and execution-time guidance within the available context. These challenges motivate our core question: *How can an agent accumulate useful experience through RSI while limiting its interference with new tasks and its recurring context cost?*

In this work, we propose *ErRsi*, a two-stage RSI framework that uses localized execution errors and their update history to guide behavioral improvement (Figure 1). In the first stage, to extract actionable error evidence within the available context, we propose elastic error localization that uses dense inspection when a trajectory fits and active investigation otherwise. The resulting diagnoses are organized into error lineages that retain failure mechanisms, occurrence conditions, and update feedback, providing compact evidence for subsequent revision. In the second stage, to translate this evidence into behavioral improvement, we introduce a revision procedure that constructs targeted updates from current diagnoses, relevant update history, and selected execution evidence. Applicability conditions control when learned guidance enters execution, while paired evaluation selects deployed changes. Together, these designs connect the evidence retained for learning with the scope and cost of applying experience during execution. Our main contributions are summarized as follows:

- We introduce ErRsi, a two-stage RSI framework that uses localized execution errors and error lineages to guide behavioral updates while keeping model weights fixed.
- We introduce elastic error localization that adapts evidence acquisition to context limits and combines process and decision views, outperforming *All-at-Once* (Zhang et al., 2025) and *DeepDebug* (Zhu et al., 2026c) in the matched 9B evaluation on Who&When.
- ErRsi reduces complete-protocol token consumption by 44.67–69.77% relative to adapted ACE across three model scales on WebShop.
- Experimental results show that ErRsi exhibits low interference when deploying frozen source experience across benchmarks, with target-task success remaining within one task of the initial agent across three model scales.

## 2 Related Work

**Agent self-improvement.** Existing work on fixed-weight agent adaptation spans reasoning–action interaction and feedback-driven revision, as represented by *ReAct*, Reflexion, and *Self-Refine* (Yao

et al., 2023, Shinn et al., 2023, Madaan et al., 2023), as well as the evolution of reusable skills, prompts, and contextual guidance through Voyager, *Genetic-Pareto (GEPA)*, ACE, and Dynamic Cheatsheet (Wang et al., 2024, Agrawal et al., 2026, Zhang et al., 2026b, Suzgun et al., 2026). These approaches provide complementary mechanisms for changing an agent's external behavior. ErRsi focuses on the evidence that directs such changes, connecting localized failures with update history and controlling when learned guidance influences execution. ACE serves as the main comparison because both methods accumulate experience through interaction.

**Trajectory failure attribution.** Who&When evaluates responsible agents and failure steps (Zhang et al., 2025). *FAMAS*, *ECHO*, and *StepFinder* use spectrum, hierarchical, and temporal evidence (Ge et al., 2026, Banerjee et al., 2025, Zhu et al., 2026d). *AgentDebugX*, *AgenTracer*, *CORRECT*, *AgentScope*, *RAFFLES*, and *DCFA* provide debugging, structured, or counterfactual attribution (Zhu et al., 2026c, Zhang et al., 2026a, Yu et al., 2025, Bi et al., 2026, Zhu et al., 2026a, Wang et al., 2026b). *SAFARI* uses active investigation for long traces (Zhu et al., 2026b). Our elastic locator combines context-dependent acquisition with shared-evidence comparison and dependency-supported ownership. Its structured output connects attribution to the conditions and execution responses needed for subsequent revision.

# 3 METHOD

Learning from execution history requires identifying which evidence should guide an update and when the resulting guidance should influence future decisions. To address these coupled challenges, we introduce a two-stage procedure that extracts actionable error evidence and uses it to construct behavioral updates with explicit application conditions. This design separates the history retained for learning from the guidance used during execution.

Let $\pi_\theta$ be a fixed-weight actor and $H_t$ its external behavior state, comprising instructions, execution rules, and persistent task knowledge. An execution on task $x$ yields a trajectory $\tau = (e_1, \ldots, e_n)$ of indexed actions, observations, and tool events, together with a binary outcome $y(x, H_t)$. The persistent record $S_t$ connects diagnoses, recurring errors, behavior versions, and update feedback. At iteration $t$, Stage 1 diagnoses failed trajectories in the observed batch $\mathcal{T}_t$ and returns the diagnosis set $D_t$. The two stages operate as follows:

$$D_t = \mathrm{Diagnose}(\mathcal{T}_t, C), \qquad \widetilde{H}_{t+1} = \mathrm{Revise}(H_t, D_t, S_t), \qquad \theta_{t+1} = \theta_t. \tag{1}$$

Here $C$ is the analysis context capacity. Paired execution selects the deployed state $H_{t+1}$ and updates $S_t$. Figure 2 presents the loop. Appendix E specifies its algorithm and execution invariants.

## 3.1 STAGE 1: ELASTIC ERROR LOCALIZATION AND LINEAGE CONSTRUCTION

Error localization must identify the decision responsible for failure from trajectories of varying length under a limited context window. To address this challenge, we introduce elastic error localization, which adapts evidence acquisition to the available context and combines process and decision views for attribution. The resulting diagnoses connect to related update history through error lineages, preparing evidence for behavioral revision.

**Context-dependent evidence acquisition.** To select the acquisition strategy, the locator profiles trajectory length, roles, text sizes, and dependencies. With estimated evidence length $\widehat{L}(\tau)$, context-dependent budget $B_C$, and dense length limit $n_{\max}$, it selects

$$E(\tau) = \begin{cases} \mathrm{Dense}(\tau), & \widehat{L}(\tau) \le B_C \text{ and } |\tau| \le n_{\max}, \\ \mathrm{Active}(\tau, B_q, B_r, B_e), & \text{otherwise.} \end{cases} \tag{2}$$

The budgets $B_q, B_r, B_e$ limit investigation rounds, reads, and acquired evidence. Every issued request additionally satisfies $\mathrm{Tok}(\mathrm{input}) + R \le C$, where $R$ reserves output capacity and a safety margin. Routing uses a size estimate, while the tokenizer enforces the actual context limit.

**Dense localization.** When the trace fits, we use dense acquisition to connect initiating decisions with their downstream consequences. The locator reads all events with their original coordinates, constructs a dependency graph, and inspects a backward slice from terminal failure events. Recovery links distinguish deviations corrected during execution from errors that persist, supporting attribution to the responsible upstream decision.

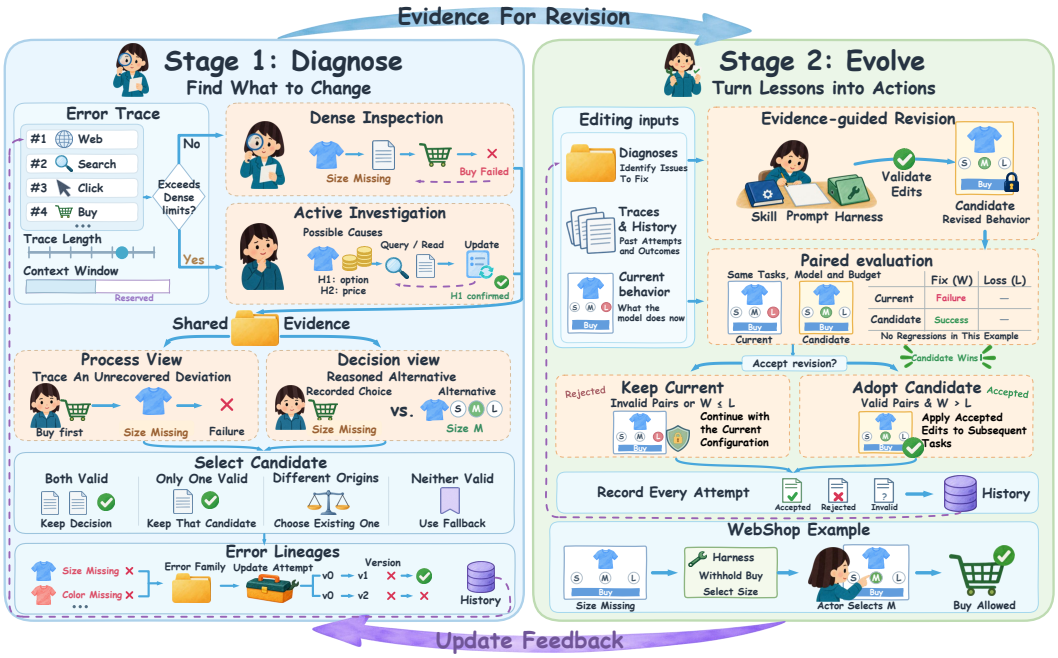


Figure 2: ErRsi links elastic error localization and error lineages to behavioral revision. Stage 1 compares process and decision views. Stage 2 tests revisions and records update outcomes.

**Active localization.** When the trajectory exceeds the available context, we use an active investigator to acquire evidence incrementally within the acquisition budget. Let $E_k$ denote acquired evidence, $Z_k$ the candidate explanations of the failure, and $U_k$ the unread intervals after round $k$. Starting from task and trace metadata, each round extends the evidence as follows:

$$g_k = \mathrm{Investigate}(E_k, Z_k, U_k), \qquad E_{k+1} = E_k \cup \mathrm{Read}_\tau(g_k),$$
$$N_{\mathrm{round}} \le B_q, \qquad N_{\mathrm{read}} \le B_r, \qquad \sum_{k=0}^{N_{\mathrm{round}}-1} \widehat{\ell}\big(\mathrm{Read}_\tau(g_k)\big) \le B_e. \tag{3}$$

Here $g_k$ is the investigation query, $N_{\mathrm{round}}$ is the total number of rounds, $N_{\mathrm{read}}$ is the total number of reads, and $\widehat{\ell}$ is the acquisition token estimate. Available queries read windows or selected steps, search events, trace dependencies, or compare states. Each query states which candidate explanations its possible results distinguish. The returned evidence updates $Z_k$ and $U_k$. Investigation stops at evidence sufficiency or an exhausted budget. Read results preserve source coordinates and explicit omission markers. Dense and active acquisition therefore supply a common evidence interface to the attribution procedure.

**Selecting the responsible decision.** A visible failure can occur downstream of the decision that initiates it. To distinguish the origin from its consequences, we compare two complementary proposals using the same evidence $E$. The process proposal $q_P$ traces propagation and recovery to locate the earliest supported, unrecovered deviation contributing to failure. Independently, the decision proposal $q_D$ identifies the responsible agent and assesses whether a locally correct replacement for a candidate decision can interrupt the failure chain. This assessment uses recorded evidence and task requirements without trajectory replay.

Let $V_E \subseteq \{q_P, q_D\}$ contain proposals whose coordinates and evidence references pass validation, and let $o(q)$ denote the proposed error origin. We select the diagnosis as follows:

$$\widehat{q} = \begin{cases} q_D, & V_E = \{q_P, q_D\},\ o(q_P) = o(q_D), \\ \mathrm{Arbitrate}(V_E, E) \in V_E, & V_E = \{q_P, q_D\},\ o(q_P) \neq o(q_D), \\ q, & V_E = \{q\}, \\ \mathrm{Fallback}(E), & V_E = \varnothing. \end{cases} \tag{4}$$

When valid proposals disagree, the adjudicator selects one of the existing candidates. If the selected event is a tool execution, we attribute it to its unique inspected non-tool initiator when an explicit

dependency supports this assignment, while retaining the tool result as propagation evidence. Each diagnosis records the error location, responsible agent, and failure mechanism, supported by the observed evidence, propagation, and recovery history. Supporting and contradicting events remain linked to the diagnosis. These diagnoses form $D_t$, the input to subsequent behavioral revision.

**Organizing error lineages.** A diagnosis identifies what goes wrong in one execution, while related update history records how previous attempts to address an error affect behavior. To make this experience available to subsequent revisions, we organize diagnoses into error lineages that connect recurring failure mechanisms with occurrence conditions and observed update responses. A model abstracts each diagnosis into a fixed normalization vocabulary covering the violated requirement, occurrence condition, propagation pattern, and recovery status. Diagnoses sharing the same violated requirement and propagation pattern belong to the same error family, while their conditions and source evidence remain attached to individual observations. For example, failures across different tasks can share the error of taking a consequential action without checking its required preconditions.

Within each error family, diagnoses remain linked to their update attempts and observed execution responses. When repeated updates produce different outcomes across occurrence conditions, the lineage can retain separate branches for those conditions. These records form $S_t$, while full trajectories remain referenced for provenance.

### 3.2 Stage 2: Behavioral Revision from Diagnoses and Update Feedback

Identifying an erroneous decision does not by itself determine how the agent should change its behavior. To address this gap, we introduce a revision procedure that constructs targeted changes to external behavior, controls when learned guidance applies, and selects updates through paired execution. Model weights remain fixed.

**Constructing revisions from error evidence.** To ground each candidate in the error it addresses, the editor receives a bounded context containing the current diagnoses $D_t$, relevant update history in $S_t$, and selected trajectory evidence. Diagnoses specify erroneous decisions, failure mechanisms, and occurrence conditions. Local actions and observations support their interpretation, previous update responses describe the effects of earlier changes, and successful executions identify behavior to preserve.

Building on this evidence, the editor constructs a candidate through the agent's supported operations for revising instructions, task procedures, and execution checks. The interface defines the available edit locations and allowed operations, while diagnoses and update feedback inform the content of the change and whether existing guidance should be retained, revised, or removed. Before evaluation, the candidate is checked for valid structure, supported operations, and configured edit limits.

**Controlling when learned guidance is applied.** However, guidance derived from a particular failure is useful only when its application conditions hold. To control its influence on later decisions, we associate learned guidance with conditions evaluated against the observable execution state and proposed action. An executable check fires only when its condition is satisfied and its firing allowance remains available. It then supplies corrective guidance and relevant observed facts before the actor chooses its next action within the original task budget. Otherwise, the check adds no guidance. Similarly, task procedures are selected through the agent's existing task interface. The implementation details specify the editable components and execution limits.

**Paired evaluation and feedback.** To determine whether a candidate improves task performance, we freeze it and evaluate it alongside the current behavior on the same task batch $\mathcal{B}_t$ under matched execution settings. For a valid paired evaluation, the change in task success and the deployment decision are

$$\Delta_t = \frac{1}{|\mathcal{B}_t|} \sum_{x \in \mathcal{B}_t} \left[ y(x, \widetilde{H}_{t+1}) - y(x, H_t) \right], \qquad H_{t+1} = \begin{cases} \widetilde{H}_{t+1}, & \Delta_t > 0, \\ H_t, & \text{otherwise.} \end{cases} \tag{5}$$

Here, $y(x, H)$ indicates whether behavior $H$ succeeds on task $x$. A candidate replaces the current behavior only when the paired comparison shows a strictly positive improvement. Ties and invalid comparisons retain the current behavior.

After paired evaluation, we append the attempted revision and its execution responses to $S_t$, linking them to the originating diagnoses. Each entry records the behavior versions, edited component,

task outcomes, and available intervention information. Both adopted and rejected candidates contribute evidence about successful updates, persistent errors, and disruptions to previously successful behavior.

# 4 EXPERIMENTS

## 4.1 EXPERIMENTAL SETUP

We first evaluate our error-localization method independently on Who&When (Zhang et al., 2025), where it achieves competitive performance against established localization approaches. We then evaluate RSI on ALFWorld (Shridhar et al., 2021) and WebShop (Yao et al., 2022), examining improvements in task success and token efficiency through learning from errors. Further experiments analyze the execution of learned guidance and the interference from accumulated experience when agents move across benchmarks. We use *Qwen3.5-4B* and *Qwen3.5-9B* (Qwen Team, 2026a), as well as *Qwen3.8-27B-FP8* (Qwen Team, 2026b), abbreviated 4B, 9B, and 27B. Actor weights remain fixed, with matched model and resource settings within each paired study.

**RSI protocol.** WebShop uses three disjoint blocks, A, B, and C, with 192 tasks each. A supplies shared initial executions under $H_0$, so both methods have the same A-block score before any behavioral update. This experience produces the first candidate $H_1$. B evaluates $H_0$ and $H_1$, and the selected version's trajectories generate $H_2$. C then evaluates both the B-selected version and $H_2$. *C incoming* is the success count on C of the version selected on B and fixed before C evaluation, measuring performance on subsequent tasks before their outcomes influence selection. *C selected* is the success count of the version retained after the paired comparison on C, which can preserve the incumbent or adopt $H_2$. The latter uses C outcomes for version selection and therefore measures performance on the selection block. Figure 3 shows the shared initial result on A and the selected scores on B and C. There is no further learning from C. Each task permits 49 decisions, a 32K actor context, and at most 100 output tokens per actor call. WebShop accounts for tokens through a shared execution ledger.

**Comparison and accounting.** Our ACE (ICLR 2026) (Zhang et al., 2026b) adaptation retains reflection and incremental playbook curation from the official implementation, with an 8,192-token playbook capacity and the same public action interface. Both methods receive experience from the same task blocks. Shared initial executions are fully attributed to each method. Complete-protocol cost includes actor execution, learning calls, and candidate evaluation, including candidates not selected for deployment. For method $m$, let $\mathcal{Q}_m$ be the multiset of logical model-call indices in this protocol, including fully attributed reused calls. Let $n_i^{\text{in}}$ and $n_i^{\text{out}}$ be the input and output token counts of call $i$. Total cost and the reported reduction are

$$C_m = \sum_{i \in \mathcal{Q}_m} \left( n_i^{\text{in}} + n_i^{\text{out}} \right), \qquad \text{Reduction} = 100 \left( 1 - \frac{C_{\text{ErRsi}}}{C_{\text{ACE}}} \right) \%. \tag{6}$$

Paired resampling summarizes variation over the evaluated tasks. Appendix A provides seeds, implementation settings, and source identities.

## 4.2 STANDALONE ERROR LOCALIZATION ON WHO&WHEN

Who&When contains 184 labeled trajectories. We report results over the complete benchmark and measure exact agreement with the annotated failure step. Our locator uses a 16K analysis context and receives no reference solution. Under the same local 9B interface, ErRsi achieves 56 exact matches (30.43%), compared with 28 for All-at-Once (15.22%) and 37 for DeepDebug (20.11%). The gains are 15.22 and 10.33 percentage points. The 27B locator reaches 68 exact matches (36.96%).

The diagnoses identify the responsible agent in 88 of 184 cases at 9B and 108 at 27B (47.83% and 58.70%). The local 9B All-at-Once and DeepDebug baselines identify 69 and 68 agents correctly, respectively. Joint agent-and-step predictions are correct in 56 and 67 cases for ErRsi at 9B and 27B, respectively. Conditional on an exact step, ownership is correct in 56 of 56 cases at 9B and 67 of 68 at 27B.

With shared acquired evidence on all 184 trajectories, combining the two views achieves 30.43% exact-step accuracy at 9B, versus 22.83% for process-only and 28.80% for decision-only attribution.

Table 1: Who&When full-benchmark exact-step accuracy (%). Venue identifies method publication. The first four published scores are from AgentDebugX. R: reference solution, †: weighted aggregate, ‡: method adapted to our task setting (16K, no reference solutions).

| Method | Venue | Configuration | Step |
|---|---|---|---|
| *Published configurations* | | | |
| All-at-Once (Zhu et al., 2026c) | ICML 2025 | Qwen3.5-9B, R | 22.30 |
| Step-by-Step (Zhu et al., 2026c) | ICML 2025 | Qwen3.5-9B, R | 18.50 |
| Binary-Search (Zhu et al., 2026c) | ICML 2025 | Qwen3.5-9B, R | 17.40 |
| DeepDebug (Zhu et al., 2026c) | EMNLP 2026 Demo | Qwen3.5-9B, R | 28.80 |
| FAMAS (Ge et al., 2026) | FSE 2026 | Qwen2.5-72B | 29.35 |
| StepFinder (Zhu et al., 2026d) | KDD 2026 | Trained, 0.6B encoder | 27.54† |
| ECHO (Banerjee et al., 2025) | NeurIPS 2025 Workshop | Claude 3.5 Sonnet v2, R | 28.58† |
| AgenTracer (Zhang et al., 2026a) | ICLR 2026 | Trained Qwen3-8B, R | 35.87† |
| AgentScope (Bi et al., 2026) | | DeepSeek-V3.2, R | 33.70† |
| RAFFLES (Zhu et al., 2026a) | EACL 2026 | Llama-3.3-70B, $K = 2$ | 36.41† |
| DCFA (Wang et al., 2026b) | | DeepSeek-R1-32B | 35.32† |
| *Local evaluation* | | | |
| All-at-Once‡ (Zhang et al., 2025) | ICML 2025 | Qwen3.5-9B | 15.22 |
| DeepDebug‡ (Zhu et al., 2026c) | EMNLP 2026 Demo | Qwen3.5-9B | 20.11 |
| ErRsi (ours) | | Qwen3.5-9B | 30.43 |
| ErRsi (ours) | | Qwen3.8-27B | 36.96 |

Agent accuracy also rises to 47.83%, versus 41.85% and 44.02%, respectively, supporting their complementary roles. Appendix C.1 details the ablation and baseline configurations.

In Table 1, our 9B accuracy exceeds the listed same-backbone prompting and DeepDebug results, plus FAMAS, StepFinder, and ECHO. 27B exceeds the listed AgenTracer, AgentScope, RAFFLES, and DCFA configurations. These published rows retain their original backbones, training, and reference inputs. The local comparison uses a matched interface. Appendix C provides additional full-benchmark configurations and aggregation details.

### 4.3 Self-Evolution on ALFWorld

ALFWorld evaluates experience organization within another environment: instructions, task-type skills, and execution configuration are editable, and the existing router selects the skill for each task type. Learning uses A and B batches. Final evaluation uses 1,184 C tasks and the official 134-task unseen split (Table 2). At 9B, successes rise from 970 to 1,002 on C (2.70 percentage points), while actor tokens fall by 27.16%. On the unseen split, the 9B actor solves 120 of 134 tasks with 34.09% fewer actor tokens.

Table 2: ALFWorld successes and actor tokens before and after RSI. C and unseen tasks are excluded from learning and selection.

| Model | Evaluation set | Successes | | Actor tokens (M) | |
|---|---|---|---|---|---|
| | | Initial | Final | Initial | Final |
| 9B | C (1,184) | 970 | 1,002 | 62.49 | 45.52 |
| 27B | C (1,184) | 1,110 | 1,149 | 40.50 | 38.60 |
| 9B | Unseen (134) | 119 | 120 | 6.34 | 4.18 |
| 27B | Unseen (134) | 128 | 134 | 4.39 | 3.93 |

At 27B, C-block successes increase from 1,110 to 1,149, a gain of 3.29 percentage points and a 52.70% net reduction in remaining failures, while actor tokens fall by 4.68%. On the unseen split, the final behavior corrects all six initial failures while preserving all 128 initial successes, solving all 134 tasks with 10.43% fewer actor tokens.

Error-guided revision therefore extends to task-type procedural knowledge and execution settings while reducing the measured deployment cost. Here cost covers evaluation-time actor execution. WebShop measures the complete evolution protocol.

### 4.4 WebShop Self-Evolution and Token Efficiency

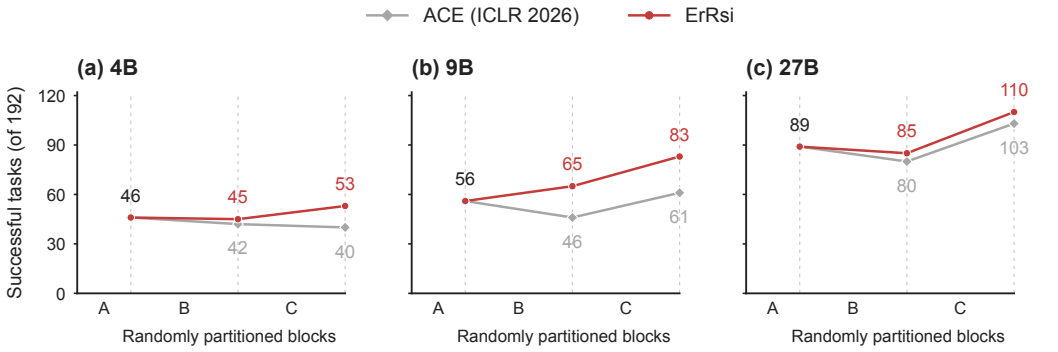


Figure 3: WebShop at 4B, 9B, and 27B on three 192-task blocks. A shows shared initial execution, while B and C show selected scores. At 9B, C incoming and C selected coincide for both methods.

Figure 3 shows the block-level comparison. At 9B, ErRsi increases B-block success from the initial 46 to 65 tasks. C incoming is 83 for ErRsi and 61 for ACE, a gain of 22 tasks or 11.46 percentage points. Neither method accepts its C-stage candidate, so these scores also equal C selected. The paired comparison contains 22 tasks solved only by ErRsi and none solved only by ACE.

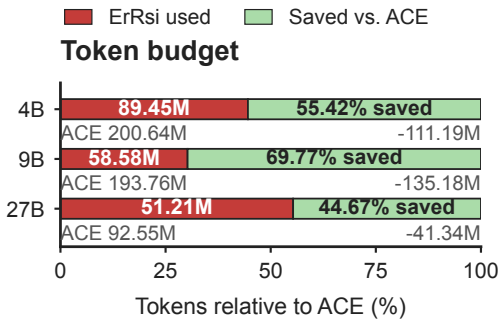


Figure 4: WebShop complete-protocol token consumption and savings relative to ACE at three model scales.

For the recorded runs, the complete 9B protocol uses 58.58M tokens for ErRsi and 193.76M for ACE, saving 135.18M tokens, or 69.77%. Figure 4 extends this comparison across model configurations: ErRsi uses 89.45M versus 200.64M tokens at 4B and 51.21M versus 92.55M at 27B, reductions of 55.42% and 44.67%, respectively. At 9B, these savings accompany the higher C incoming success reported above. To account for differences in experience representation, we also evaluate a smaller, compressed version of the frozen 9B ACE playbook on 32 WebShop tasks. ErRsi solves ten tasks versus eight for compressed ACE while using 90.92% fewer actor tokens (Appendix F.1).

At 4B, Figure 3 shows 45 versus 42 successes on B and 53 versus 40 on C for ErRsi and ACE, respectively. The C-block difference is 13 tasks, or 6.77 percentage points. At 27B, the figure shows 85 versus 80 on B and 110 versus 103 on C, with a C-block difference of seven tasks, or 3.65 percentage points. Appendix D reports the separate follow-up increasing only the 4B analysis context.

**Diagnostic evidence for revision.** For the first 9B revision, we hold the A-block experience, editor, and editing interface fixed. Diagnoses and selected evidence solve 65 of 192 B tasks versus 47 with raw trajectory input (+9.38 percentage points), reducing actor tokens from 33.28M to 7.89M (76.30%). The raw control removes Stage 1 before evidence selection and window construction. Removing only diagnostic text from the same selected evidence yields 63 successes and 10.78M tokens, a 26.79% actor-token saving with diagnostics. Appendix A.4 details both controls.

### 4.5 Executing Learned Knowledge and Preserving Capabilities

**Execution of the same learned content.** We hold WebShop knowledge fixed and compare textual and executable presentations on 384 tasks, isolating how stored experience affects later decisions. The same 9B-learned content is applied to 4B, 9B, and 27B actors. Both arms share model responses until their contexts or actions diverge, and each is charged its full logical tokens. No additional learning occurs in this comparison.

Figure 5 shows 158 to 180 successes at 4B and 154 to 167 at 9B, gains of 5.73 and 3.39 percentage points. The paired results contain 22 fixes at 4B and 13 at 9B, with no regressions in either experi-

ment. The 95% paired bootstrap intervals for the gains are [3.39, 8.07] and [1.82, 5.21] percentage points. The 27B actor solves 211 tasks in each form. Actor-token totals and intervention counts appear in Appendix B.

The program intervenes on 44 tasks at 4B and 34 at 9B (11.46% and 8.85%), with 44 and 35 blocked decisions. At 27B, 13 tasks receive 13 interventions. Since learned content is fixed, these counts reflect when each actor meets the checks' execution conditions. The 4B and 9B gains show how experience can improve aggregate success through a small number of targeted interventions, preserving the distinction between storing a lesson and applying it to every task. Because the knowledge is acquired with a 9B actor, its benefit on 4B also demonstrates reuse across model scales. The smaller actor benefits from previously learned updates without repeating the learning process.

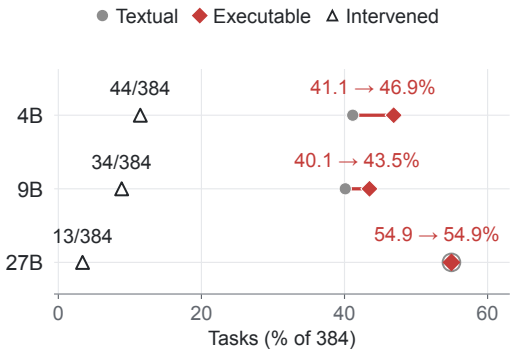


Figure 5: Success and intervention rates for the same 9B-learned knowledge on 384 WebShop tasks per actor.

**Cross-benchmark performance retention.** We compare frozen WebShop versions of ErRsi and ACE on all 134 ALFWorld unseen tasks. Each method uses its selected source version. If no evolved version is automatically accepted, we choose the generated candidate with the highest recorded WebShop success rate, excluding $H_0$. Selection uses only source-domain results, with no target learning or selection. Each target task has a 32K context, 49 decisions, and 400K tokens. The 4B ErRsi source comes from the 64K-analysis follow-up. Its actor context remains 32K. Figure 6 includes initial behavior $H_0$ as a reference.

Across the 4B, 9B, and 27B models, ErRsi solves 86, 112, and 125 target tasks, compared with 61, 107, and 116 for ACE and 87, 113, and 126 for the initial agent. Under the same target limits, ErRsi stays within one task of initial success at all three model scales while solving more tasks than ACE. Its target-execution token consumption is 76.19%, 75.39%, and 78.64% lower than ACE, respectively.

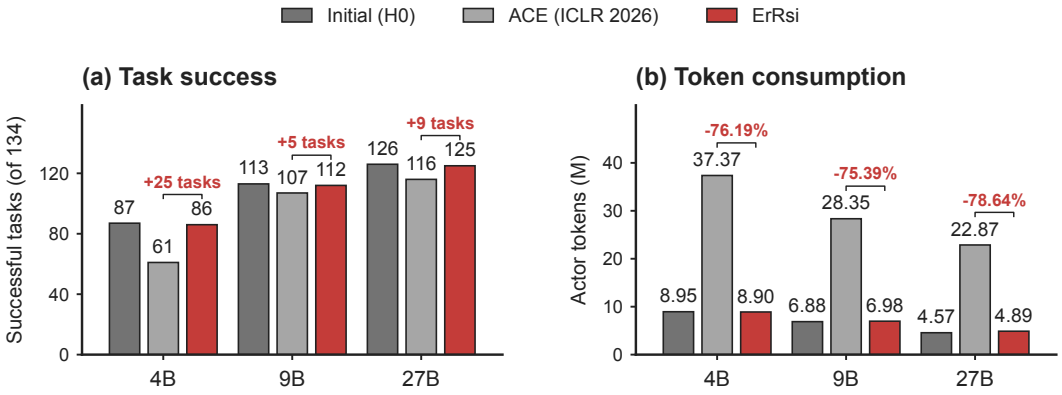


Figure 6: WebShop-to-ALFWorld transfer on 134 unseen tasks using selected source versions. Left: successes relative to $H_0$. Right: target-execution tokens. Target limits match across methods.

## 5 Conclusion

Agent self-evolution must make accumulated experience useful across changing tasks while controlling its recurring cost. ErRsi combines elastic error localization, compact lineage records, and bounded behavioral revision. Evaluations connect accurate localization to improved source-task execution, lower token consumption, and little interference from past experience after a benchmark change. Specific errors and observed update feedback support continued learning without placing the entire experience history in every decision.

## AI USE STATEMENT

Generative AI tools assist method development, implementation, experiment design and execution, literature retrieval, result analysis, manuscript drafting and editing, and figure preparation.

## REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Adi Banerjee, Anirudh Nair, and Tarik Borogovac. Where did it all go wrong? a hierarchical look into multi-agent error attribution. *arXiv preprint arXiv:2510.04886*, 2025.

Jiayi Bi, Yanjie Gao, Yuanmin Xie, Liqun Li, Tianyin Xu, Fan Yang, and Mao Yang. Diagnosing with insights: Structured analysis of agent failures via behavioral abstractions. *arXiv preprint arXiv:2609.02371*, 2026.

Yu Ge, Linna Xie, Zhong Li, Yu Pei, and Tian Zhang. Spectrum-based failure attribution for multi-agent systems. *Proceedings of the ACM on Software Engineering*, 3(FSE):1888–1909, 2026.

Kangsan Kim, Minki Kang, Taeil Kim, Yanlai Yang, Mengye Ren, and Sung Ju Hwang. Memory transfer learning: How memories are transferred across domains in coding agents. *arXiv preprint arXiv:2604.14004*, 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. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026a. URL https://qwen.ai/blog?id=qwen3.5.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL https://qwen.ai/blog?id=qwen3.8.

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

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

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Yuyao Wang, Zhongjian Zhang, Mo Chi, Kaichi Yu, Yuhan Li, Miao Peng, Bing Tong, Chen Zhang, Yan Zhou, and Jia Li. Evomembench: Benchmarking agent memory from a self-evolving perspective. *arXiv preprint arXiv:2605.18421*, 2026a.

Zehao Wang, Lanjun Wang, Shilong Jin, Junjie Chen, and Yanghua Xiao. Dcfa: Dual-view causal-inspired attribution for failure reasoning in llm-based multi-agent systems. *arXiv preprint arXiv:2609.04749*, 2026b.

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

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

Yifan Yu, Moyan Li, Shaoyuan Xu, Jinmiao Fu, Xinhai Hou, Fan Lai, and Bryan Wang. Correct: Condensed error recognition via knowledge transfer in multi-agent systems. *arXiv preprint arXiv:2509.24088*, 2025.

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, 2026a.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026b.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. In *International Conference on Machine Learning*, pp. 76583–76599. PMLR, 2025.

Chenyang Zhu, Spencer Hong, Jingyu Wu, Kushal Chawla, Yuhui Tang, Youbing Yin, Nathan Wolfe, Erin Babinsky, and Daben Liu. Raffles: Reasoning-based attribution of faults for llm systems. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7659–7688, 2026a.

Chenyang Zhu, Jiayu Yao, Kushal Chawla, Youbing Yin, Nathan Wolfe, Pengshan Cai, Jingyu Wu, Spencer Hong, Sangwoo Cho, Shi-Xiong Zhang, et al. Safari: Scaling long horizon agentic fault attribution via active investigation. *arXiv preprint arXiv:2606.24626*, 2026b.

Kunlun Zhu, Xuyan Ye, Zhiguang Han, Yuchen Zhao, Bingxuan Li, Weijia Zhang, Muxin Tian, Xiangru Tang, Pan Lu, James Zou, et al. Agentdebugx: An open-source toolkit for failure observability, attribution, and recovery in llm agents. *arXiv preprint arXiv:2607.18754*, 2026c.

Taiyu Zhu, Yifan Wu, Weilin Jin, Ying Li, and Gang Huang. Stepfinder: A temporal semantic framework for failure attribution in multi-agent systems. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 6946–6957, 2026d.

# A Experimental Protocols and Implementation

## A.1 WebShop Rolling Evaluation

Each of A, B, and C contains 192 disjoint tasks. A executes the initial actor and produces the first update. On B, both the initial behavior and the first candidate execute. Strictly positive paired net success selects the candidate. The selected B trajectories supply the second update. C executes its incoming behavior and this second candidate. Selection uses valid paired runs and the same strictly positive net-success rule. Ties preserve the incumbent. C incoming evaluates the version selected on B before C-based selection. C selected reports the score of the version retained after the comparison on C. The reported main run uses actor seed 20260908 and editor seed 20260905. Two additional 9B replications use actor/editor seed pairs (20261008, 20261005) and (20270108, 20270105).

Each compared actor uses a 32,768-token context, at most 49 decisions, and at most 100 output tokens per call. WebShop uses a shared token ledger without enforcing a per-task token cap. The 400,000-token cap applies to ALFWorld transfer. The complete protocol includes shared initial experience, learning, and all candidate evaluations. ACE reuses 384 initial A/B episodes where available. Their full logical cost is added to its attributed total. The ACE (Zhang et al., 2026b) adaptation is based on official revision `82709de0`, with an 8,192-token playbook, reflection without reference answers, and sequential incremental ADD curation. The actor receives the full playbook at each decision under the shared public action contract.

Complete-protocol totals are 58,582,872 tokens for ErRsi and 193,762,541 for ACE at 9B, and 51,209,559 versus 92,551,270 at 27B. For 4B, ErRsi uses 89,453,261 total tokens. ACE uses 165,508,744 newly executed tokens plus 35,134,828 attributed initial tokens, totaling 200,643,572.

## A.2 Stage 1 Acquisition and Attribution

Dense routing uses an initial evidence budget of $0.6C$ and a maximum of 120 steps. Tokenizer-based validation reserves instructions and output space before issuing the request. Formal 16K standalone localization bounds active investigation to eight rounds, eight reads, and 10,000 estimated acquired-evidence tokens. The investigator maintains up to four candidate explanations. Reading operations preserve source indices. Oversized results carry omission markers. If an initial query is empty or the investigator attempts to stop before obtaining evidence, the implementation reads a terminal anchor.

Both process and decision analyses use the same acquired evidence. Validity checks enforce coordinates and references. Agreement retains the decision proposal. Two valid but different origins invoke an adjudication step over both candidates. One valid candidate is selected directly. Two invalid candidates invoke the implementation's deterministic fallback. Tool ownership is projected only to a unique inspected non-tool parent with a supported dependency beyond temporal adjacency. The alternative action in the decision analysis is generated by reasoning over evidence, not by an environment replay. Standalone evaluation supplies neither human attribution labels nor reference solutions to the locator.

## A.3 Stage 2 Editable Components

In WebShop, Stage 2 learns execution-rule updates from Stage 1 diagnoses and recorded execution feedback. Its editor receives the current behavior, rule applicability, prior update responses, and selected public trajectory evidence. Up to eight failed and four successful trajectories are selected with task-group diversity. Failed purchases receive priority. Condensed diagnoses retain the origin, mechanism, supporting and contradicting steps, and recovery status.

The execution interface does not supply fixed trigger logic: the editor itself writes each check's code, deciding how the check evaluates the observable state and the proposed action and thus when learned guidance applies. Which checks to combine, their corrective guidance, and their firing limits are likewise determined by the editor from the supplied evidence. A candidate contains zero to four checks, with 10–400 characters of guidance and a firing limit of one or two per check. A task permits at most four withheld actions, each consuming the original decision budget. Checks use public observations, and the actor chooses its next action after receiving guidance.

Opportunity counts summarize applicable successful and failed task groups. Historical response fragments describe the parent program's rules and supply available examples of fixes, regressions, persistent failures, and retained successes, together with earlier attempts from that parent. Schema correction permits at most two drafts in total, including the initial draft, and does not run additional environment evaluations.

The ALFWorld study exposes instructions up to 2,400 characters, task-type skills, and execution settings. Each skill has five fields: objective, procedure, state checks, recovery, and budget. Fields are 20–700 characters, and the objective is protected after initialization. The existing task-type router selects the skill. Execution edits select supported settings for mode, history, feedback, repeated-action handling, or action syntax. Each learning window has up to three candidate slots, with a skill revision first. Selection uses A and then cumulative A+B execution. Final evaluation uses C and the official unseen split. Those evaluation tasks are excluded from the run's learning and selection.

### A.4 Diagnostic-Input Controls

Both controls use the 9B first revision learned from the same A192 initial experience and evaluated on all 192 B tasks. The diagnosed reference is the frozen $H_1$ used in the main study. Each condition produces one candidate using the same editing interface and model settings.

**Raw trajectory input.** The raw-input control removes Stage 1 before evidence selection and local-window construction, allowing the editor to learn from public trajectory evidence without the diagnostic input path. Acquisition remains subject to the available context. Raw input does not mean concatenating every complete trajectory into one request. The diagnosed candidate solves 65 tasks with 7,888,195 actor tokens, versus 47 with 33,276,714 for the raw-input candidate. Paired outcomes contain 33 diagnosis-only and 15 raw-only successes. The net gain is 18 tasks, or 9.38 percentage points, with 76.30% fewer actor tokens. The reported execution reduction excludes editing, diagnosis generation, and the shared A-block experience. After inspecting the initial 64-task evaluation, we extend it to B192 without generating additional candidates or seeds.

**Removing only diagnostic text.** This control removes the diagnostic field from the same twelve detailed examples while retaining their public states and events, evidence selection, the operation interface description, summary statistics, parent program, and editor instructions. It therefore tests explanatory text conditional on the original selected evidence. We specify the comparison after the reference B results become available.

With diagnostics, the learned program solves 65 tasks using 7,888,195 actor tokens. Without diagnostics, it solves 63 using 10,775,096 tokens. The diagnosed condition has two paired fixes and no regressions, with a 26.79% actor-token reduction. Costs exclude diagnosis generation and editing. Both arms are charged their full logical execution cost, including shared recorded responses before the first request divergence. Subsequent control requests use new inference.

## B Additional Execution and Transfer Results

Table 3: Fixed-knowledge execution on 384 WebShop tasks. Both arms are charged their full logical actor tokens, including shared responses before divergence. Fixes and regressions are paired relative to the textual arm.

| | Successes | | | | Actor tokens | |
|---|---|---|---|---|---|---|
| Model | Text | Executable | Fixes | Regressions | Text | Executable |
| 4B | 158 | 180 | 22 | 0 | 13,022,364 | 13,251,955 |
| 9B | 154 | 167 | 13 | 0 | 14,901,655 | 15,973,946 |
| 27B | 211 | 211 | 0 | 0 | 9,546,652 | 9,919,834 |

The 4B run intervenes on 44 tasks with 44 blocked decisions. The 9B run intervenes on 34 tasks with 35 blocked decisions. Exact two-sided McNemar probabilities for the 4B and 9B paired successes are $4.77 \times 10^{-7}$ and $2.44 \times 10^{-4}$. Their paired task-bootstrap intervals are reported in the main text. These task-level comparisons use one actor seed and the same 9B-learned content across actor configurations.

### B.1 Cross-Benchmark Transfer of Source Knowledge

The target manifest contains 134 official unseen tasks: 18 examine-in-light, 24 simple placement, 31 cleaning, 21 cooling, 23 heating, and 17 two-object placement tasks, distributed across four room groups. Each distinct frozen configuration executes each task once with actor seed 20260908. The target allows 49 decisions, a 400,000-token total budget, at most 100 output tokens per call, and a 32,768-token context. There is no target-domain diagnosis, editing, or selection. These tasks are absent from the source WebShop learning runs.

Table 4: WebShop-to-ALFWorld transfer on 134 tasks per model. Fixes/losses are relative to $H_0$. Tokens cover target execution. * marks 9B ACE, which does not pass source acceptance. ACE is adapted from ICLR 2026.

| Model | Version | Successes | Fixes | Losses | Actor tokens |
|---|---|---|---|---|---|
| 4B | $H_0$ | 87 | – | – | 8,947,693 |
| | ACE | 61 | 18 | 44 | 37,366,785 |
| | ErRsi | 86 | 8 | 9 | 8,897,132 |
| 9B | $H_0$ | 113 | – | – | 6,879,471 |
| | ACE* | 107 | 12 | 18 | 28,345,819 |
| | ErRsi | 112 | 4 | 5 | 6,976,949 |
| 27B | $H_0$ | 126 | – | – | 4,566,477 |
| | ACE | 116 | 3 | 13 | 22,873,815 |
| | ErRsi | 125 | 4 | 5 | 4,885,192 |

The ErRsi version prefixes are `10ab9cb35621`, `1c22ff4c5a32`, and `38f435f4f87e` for 4B, 9B, and 27B. ACE uses `c628f6acfefe`, `41bdd33d6275`, and `fde50e3adc84`. When no source update is automatically accepted, we rank all generated candidates from that source run by their recorded WebShop success rates and select the highest-scoring candidate rather than $H_0$. For 9B ACE, this selects the second candidate, `41bdd33d6275` (Table 5). The scores come from each candidate's registered source block, rather than a new common validation set. The version is fixed before target evaluation, and target outcomes do not enter this choice.

Table 5: 9B ACE source candidates, evaluated on their respective 192-task WebShop blocks. Neither passes paired acceptance. The higher recorded candidate score determines the experience-bearing transfer version.

| Candidate | Source block | Incumbent successes | Candidate successes | Accepted |
|---|---|---|---|---|
| First | B | 46 | 34 | No |
| Second | C | 61 | 51 | No |

The 4B ErRsi transfer version comes from the 64K-analysis follow-up in Appendix D, whereas its main WebShop run uses 32K analysis. Both actors use a 32K execution context.

The supplement adds only 134 ACE executions, 2,736 requests, and 28,345,819 tokens. $H_0$ and ErRsi retain their audited records. Across all three scales, the comparison contains 1,206 distinct executions, 26,843 requests, and 129,739,333 tokens. Twelve earlier identity probes cost another 252 tokens, excluded from method totals. The 9B supplement uses two BF16 replicas and eight environment slots, versus four replicas and sixteen slots for its inherited runs. Weights, actor settings, and task order match.

ACE's 73, 27, and 18 budget stops at 4B, 9B, and 27B occur when the next complete request cannot fit the remaining total-token allowance. For all 27 failed 9B tasks, the next request fits the 32K context and the decision count remains below 49. Accumulated use is 388,366–399,891 tokens. Thus the constant playbook's budget cost contributes to this comparison, which does not separate that cost from interference caused by its content.

Direct paired comparisons yield 38 ErRsi-only versus 13 ACE-only successes at 4B, 18 versus 13 at 9B, and 12 versus 3 at 27B. The 10,000-resample room-cluster 95% intervals for these success-rate differences are $[15.22, 20.45]$, $[0.41, 13.33]$, and $[4.35, 11.67]$ percentage points, respectively. These intervals use four room groups and one actor seed under the stated target settings.

# C Localization Sources and Additional Configurations

All local results use the full 184-trajectory benchmark. The 9B locator has 56 exact-step matches and 88 correct agent identities. 56 trajectories have both correct. The 27B locator has 68 exact-step matches, 108 correct agent identities, and 67 joint matches.

When a publication provides partition-level rates covering the complete benchmark, we reconstruct overall accuracy using the original sample counts as weights. Published rounding can affect the last displayed decimal. Only statistics supporting a full-benchmark aggregate are included. Published configurations can differ in input availability, training, replay, and backbone. The named configuration is part of each reported result.

Table 6: Additional published Who&When configurations with full-benchmark exact-step accuracy (%). R denotes reference-solution input. Each citation applies to adjacent configurations of the same method.

| Method | Configuration | Step |
|---|---|---|
| FAMAS (Ge et al., 2026) | Qwen2.5-72B | 29.35 |
| StepFinder (Zhu et al., 2026d) | Trained, 0.6B encoder | 27.54 |
| ECHO (Banerjee et al., 2025) | Claude 3.5 Sonnet v2 | 27.07 |
| ECHO | Claude 3.5 Sonnet v2, R | 28.58 |
| AgenTracer (Zhang et al., 2026a) | Trained Qwen3-8B | 32.06 |
| AgenTracer | Trained Qwen3-8B, R | 35.87 |
| AgentScope (Bi et al., 2026) | DeepSeek-V3.2 | 33.15 |
| AgentScope | DeepSeek-V3.2, R | 33.70 |
| RAFFLES (Zhu et al., 2026a) | Llama-3.3-70B, $K = 2$ | 36.41 |
| DCFA (Wang et al., 2026b) | DeepSeek-R1-32B | 35.32 |
| DCFA | Qwen3-Coder-30B | 30.43 |

## C.1 Local Baselines and Attribution Components

Table 7 adds agent identity and joint agent–step results for the original frozen 9B runs. Joint correctness requires both the annotated step and responsible agent to match. The full 184-trajectory denominator is retained for all methods, including unsubmitted and invalid predictions.

Table 7: Local 9B attribution results on all 184 Who&When trajectories. Entries are correct predictions. Joint requires both the agent and exact step. ‡ denotes a method adapted to our task setting.

| Method | Exact step | Agent | Joint |
|---|---|---|---|
| All-at-Once‡ | 28 | 69 | 27 |
| DeepDebug‡ | 37 | 68 | 35 |
| ErRsi | 56 | 88 | 56 |

**Baseline configurations.** All-at-Once uses the AgentDebugX implementation of the attribution strategy. The local baselines use no reference solution, a 16K request context, a non-thinking Qwen profile, temperature 0.7, and at most 2,400 output tokens, retaining tighter stage-specific limits. The published DeepDebug configuration uses reference solutions, temperature 0, and up to 4,096 output tokens. The local results therefore measure adaptation to a shared task setting rather than reproduction of those published absolute scores.

**Process and decision views.** Table 8 compares process-only, decision-only, and combined predictions on the same 184 Who&When trajectories. All configurations reuse the same acquired evidence and cached proposals, isolating attribution from evidence acquisition.

Table 8: Process/decision-view ablation on all 184 Who&When trajectories with 9B and shared acquired evidence. Entries are correct predictions.

| Configuration | Exact step | Within one step | Agent | Joint |
|---|---|---|---|---|
| Process view only | 42 | 64 | 77 | 42 |
| Decision view only | 53 | 70 | 81 | 53 |
| ErRsi | 56 | 71 | 88 | 56 |

## D Analysis-Context Follow-up

The completed 4B follow-up keeps the actor context at 32K and expands analysis to 64K. It accepts one update on B, increasing success from 40 to 44 of 192 tasks (12 fixes and eight losses). The selected version solves 59 C tasks. The second proposal has the same executable content, so C is executed once and reused for the identical candidate. Complete cost is 76,223,403 tokens, including 35,134,828 attributed initial-experience tokens. Newly generated tokens total 41,088,575. This final version supplies the 4B cross-benchmark deployment. The main WebShop comparison retains the original 32K-analysis study.

Table 9: Completed 4B analysis-context follow-up. C incoming and selected refer to the same executed version.

| Actor context | Analysis context | C incoming | C selected | Total tokens |
|---|---|---|---|---|
| 32K | 64K | 59 | 59 | 76,223,403 |

## E Algorithm and Execution Invariants

**Algorithm 1: One diagnosis–revision–evaluation iteration.** The inputs are the fixed actor $\pi_\theta$, current external state $H_t$, lineage record $S_t$, observed trajectories $\mathcal{T}_t$, context capacity $C$, and evaluation tasks $\mathcal{B}_t$. The execution interface supplies the supported edit locations and operations.

1 For each failed trajectory, profile its size and select dense or active acquisition using Equation 2.
2 Acquire indexed evidence within the request and read budgets. Retain dependencies and recovery information.
3 Form the process and decision proposals over the same evidence. Validate and select with Equation 4, then resolve supported tool ownership.
4 Assemble diagnoses $D_t$ and record their family assignments and source observations in the lineage.
5 Construct a bounded editing context from diagnoses, public states, applicability, and relevant update feedback. Propose a revision through the supported interface and validate its structure, operations, and edit limits.
6 Freeze the candidate, then execute the incumbent and candidate on $\mathcal{B}_t$ under the registered paired protocol.
7 Compute $\Delta_t$, select $H_{t+1}$ by Equation 5, and retain the paired execution responses.
8 Append the diagnoses, attempted revision, and paired execution responses to the persistent record. Return $(H_{t+1}, S_{t+1})$.

Schema correction is performed before candidate freezing and uses no evaluation-task feedback. If a valid candidate is unavailable, the controller retains the incumbent and records the proposal outcome. WebShop instantiates the iteration with A as initial experience, B as the first paired evaluation and next source of experience, and C as the final paired evaluation. There is no proposal after C.

**Bounded corrective execution.** Each withheld action requires an eligible rule with a remaining firing allowance and a remaining global block allowance. Counters increase when the rule fires or the action is withheld. For a program with $N_{\text{check}}$ checks, let $b_j \in \{1, 2\}$ be the firing allowance of check $j$, $B_{\text{block}} = 4$ the global block allowance, and $N_{\text{block}}$ the number of withheld actions. Then

$$N_{\text{block}} \leq \min\left(B_{\text{block}}, \sum_{j=1}^{N_{\text{check}}} b_j\right) \leq 4. \tag{7}$$

Multiple rules can fire at one decision. This can only consume firing allowances faster relative to the number of withheld actions. Every such decision counts toward the original 49-decision WebShop limit. The bound controls how often learned guidance can interrupt execution within that budget.

**Empirical selection property.** For valid paired outcomes on the registered comparison block, define $\widehat{J}_{\mathcal{B}_t}(H) = |\mathcal{B}_t|^{-1} \sum_{x \in \mathcal{B}_t} y(x, H)$. Equation 5 gives

$$\widehat{J}_{\mathcal{B}_t}(H_{t+1}) = \widehat{J}_{\mathcal{B}_t}(H_t) + \max(0, \Delta_t). \tag{8}$$

This identity describes the empirical selection decision on that block. C incoming separately evaluates the behavior fixed by the earlier B selection. Figure 3 reports final selected scores, and the main text also identifies the 9B C incoming result.

**Lineage relations.** The initial family identity is determined by the violated requirement and propagation pattern. The implementation records a shared-update relation when an edit operator corrects failures in at least two distinct task groups in each of two families. It creates condition-specific children when an operator has strictly positive responses in at least two groups for one condition and nonpositive responses in at least two groups for another. These relations organize recurring observations and update histories. Context construction selects the relevant records for each revision request. Appendix A.3 specifies this evidence selection for WebShop and the historical ALFWorld experiment.

**Iteration scope.** The WebShop study evaluates two candidate updates: the first uses A experience, and the second uses the selected B trajectories and available update feedback. There is no further update or independent post-selection block after C. This protocol measures short-horizon RSI and separates candidate proposal from deployment. It does not measure a long-run accumulation curve.

# F Supplementary Revision and Deployment Controls

The following controls use frozen task sets and candidate identities. Unless explicitly stated otherwise, token totals cover target actor execution, including the full logical cost of reused responses. They are distinct from the complete-protocol totals in the main WebShop comparison.

## F.1 Compressed ACE Playbook Deployment

We compress one frozen 9B ACE source playbook from 8,169 to 1,888 tokens in a single pass without target feedback. The source is the experience-bearing candidate used in the transfer comparison. The same actor evaluates the complete and compressed playbooks, ErRsi, and $H_0$ on a fixed 32-task WebShop subset (Table 10). This changes deployment of existing knowledge. ACE is not retrained under a smaller capacity.

On WebShop, compression raises ACE success from five to eight tasks and reduces actor tokens by 35.23%. ErRsi solves ten tasks and uses 90.92% fewer actor tokens than compressed ACE. The compressed text loses some conditions and exceptions, so these results characterize this deployment variant rather than isolating text length alone.

Table 10: Frozen 9B playbook deployment on 32 WebShop tasks. The compressed ACE text has 1,888 tokens. Costs cover target actor execution.

| Behavior | Successes | Actor tokens (M) |
|---|---|---|
| $H_0$ | 8 | 0.710 |
| Full ACE | 5 | 16.890 |
| Compressed ACE | 8 | 10.940 |
| ErRsi | 10 | 0.994 |

# G Implementation and Artifact Details

**Dependency construction and fallback.** The dependency graph includes explicit parent references, consecutive writers of the same recorded state field, and links from an erroneous event to a recorded recovery attempt. A backward traversal from a terminal event restricts inspection to its recorded ancestors. For an illustrative trace, if event 5 depends on event 2 and terminal event 8 depends on event 5, the backward slice contains 2, 5, and 8. Their temporal order alone does not establish those edges. Tool ownership is assigned only when the inspected graph identifies a unique non-tool initiator.

When both attribution proposals are invalid, the deterministic fallback selects an inspected terminal event, collects its recorded ancestors, removes events with successful recovery, and chooses the earliest remaining nonterminal ancestor. If none remains, it selects the terminal event. When two valid proposals disagree, the adjudicator can return only one of their proposed origins. An invalid adjudicator choice defaults to the decision proposal. These rules supply an explicit result for incomplete or malformed model proposals.

**Lineage categories.** The abstraction uses four finite category sets (Table 11). Each assignment cites an observed event and a quotation present in its action or observation. Family identity uses the violated requirement and propagation pattern. Occurrence conditions remain attached to observations until response evidence supports a branch. The categories describe execution failures across tasks, while the behavior interface bounds which corrections an editor can express.

Table 11: Category vocabulary used to normalize diagnosis mechanisms. Each axis includes an unresolved value.

| Axis | Values |
|---|---|
| Violated requirement | Entity binding, action schema, action precondition, state update, goal evidence, progress control, resource allocation, information coverage, coordination, unresolved |
| Occurrence condition | Ambiguous reference, missing precondition, no effect, repeated action, unsupported completion, unread evidence, handoff, budget pressure, unresolved |
| Propagation | Invalid action, stale belief, wrong subgoal, repeated no progress, resource exhaustion, unresolved |
| Recovery | Unchecked, misread feedback, ineffective retry, not attempted, unresolved |

**Editor input and output.** The editor uses Stage 1 diagnoses, selected execution evidence, current behavior, and recorded update responses to propose a revised rule set through the interface described in Appendix A.3. Its output identifies the proposed update, behavior to preserve, supporting evidence, and the revised rule set. The recorded 9B first-update request uses temperature 0.7, top-$p$ 0.8, top-$k$ 20, a 4,096-token output limit, and editor seed 20260905.