BEYOND RECALL: COMPILING EXPERIENCE INTO EXECUTABLE MEMORY FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon LLM agents must turn noisy, sparse-reward trajectories into reusable experience, yet complete trajectories retain incidental actions and summaries can omit execution conditions. We introduce `Trace`, which compiles executable Walkthrough Memory with a fixed Actor: delayed credit identifies candidate preparation, cross-episode outcomes yield candidate prerequisites, and backward slicing links target facts to observed producers. Entry conditions, expected effects, and termination predicates support state-conditioned scheduling, monitoring, and resumption. Across WebShop, J-TTL, and ScienceWorld with three LLM backbones under episode resets, `Trace` outperforms eight baselines in AUC and Final-3. Relative gains over the strongest baseline in benchmark-averaged AUC and Final-3 average 30.9% and 41.6% across backbones. Ablations show complementary gains from dependency-grounded construction, executable representation, and prerequisite-aware scheduling.

1 INTRODUCTION

Large Language Model (LLM) agents tackle long-horizon tasks such as web navigation, text-based games, and tool use (Zhou et al., 2024; Yao et al., 2022; Wang et al., 2022), but partial observability, sparse feedback, and long prerequisite chains impede progress. Test-time self-evolving agents accumulate experience across episodes through external memory or prompt updates while keeping the model fixed (Shinn et al., 2023; Zhao et al., 2024; He et al., 2026). Their improvement depends on converting noisy trajectories into memories that subsequent Actors can select, execute, and verify.

For example, opening a gate may require asking a technician, searching a desk, and taking a code, although only opening the gate receives reward. The same trajectory may contain incidental navigation and repeated observations. Memory construction must distinguish useful preparation from detours, identify local procedure boundaries, and preserve execution conditions. We formulate ***Walkthrough induction from sparse-reward trajectories*** as recovering compact, state-conditioned procedures that can be monitored and resumed in subsequent episodes.

Episodic retrieval recalls experience (Huang et al., 2025; Wang et al., 2026) without itself determining replay steps or boundaries. **Reflective memory** distills insights (Shinn et al., 2023; Zhao et al., 2024; Zhang et al., 2025b), often omitting preconditions, effects, and completion criteria. **Procedural memories** extract routines (Cao et al., 2026a; Liang et al., 2026c; Sun et al., 2026; Wang et al., 2024), but generally do not test whether retained transitions establish facts used downstream. **Graph memories** organize temporal, semantic, or hierarchical relations (Zhang et al., 2026a; Ranaldi, 2026; Liu et al., 2025); execution dependencies additionally require upstream effects to satisfy downstream conditions. What remains missing is evidence that a transition enabled later progress, beyond historical association with success.

`Trace` combines two signals for Walkthrough induction: delayed credit identifies potentially useful, unrewarded preparation, while dependency evidence links observed effects to target facts or candidate prerequisites. Together they support compact action chains; neither high credit nor observational dependency evidence proves causal necessity.

`Trace` canonicalizes observations, anchors progress, and propagates credit backward. It estimates prerequisites from cross-episode outcomes and slices from anchor facts to observed producers. Retained transitions become Walkthroughs with entry conditions, prerequisites, precondition-action-

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.

effect steps, and completion/failure predicates, enabling state-conditioned scheduling, resumption, and monitoring with a fixed Actor.

We evaluate WebShop, J-TTL, and ScienceWorld with `Qwen3-32B`, `Mistral-Small-3.2-24B`, and `GPT-OSS-20B`. Ablations examine cumulative construction gains and complementary representation/scheduling effects. Progress masking, trajectory noise, and history budgets probe construction evidence; separate online perturbations test execution recovery within controlled conditions.

Our main contributions are as follows:

- **Credit-Guided, Dependency-Grounded Induction.** `Trace` combines progress anchoring and delayed credit with experience-grounded prerequisite estimation and backward slicing. It recovers unrewarded preparation from noisy, sparse-reward trajectories and removes actions unsupported by the observed candidate dependency chain.
- **Executable Self-Evolving Walkthrough Memory.** Walkthroughs encode entry conditions, candidate prerequisites, precondition-action-effect steps, and completion/failure predicates. This enables monitored, resumable execution and cross-episode memory refinement with a fixed Actor.
- **Cross-Benchmark Gains and Mechanistic Diagnostics.** `Trace` leads every evaluated benchmark-backbone block, improving benchmark-averaged AUC and Final-3 over the strongest baselines by 30.9% and 41.6%, averaged across backbones. Ablations and trajectory, perturbation, and history-efficiency diagnostics examine the roles of credit propagation, prerequisite estimation, and dependency slicing.

2 Related Work

Test-Time Learning for Self-Improving LLM Agents. Self-improvement methods build reflections (Ge et al., 2025), train policies from interactions or generated tasks (Fang et al., 2025; Qi et al., 2025; Xiao et al., 2026; Zhou et al., 2026; Zhang et al., 2026d), learn memory-update policies (Cai et al., 2026), or optimize prompts, code, and workflows (Belle et al., 2025; Liu et al., 2026a; Yin et al., 2025; Zhang et al., 2025a). Long-horizon credit assignment (Kim et al., 2026b) motivates TRACE and SALT's training signals (Tao et al., 2026; Li et al., 2026a); our credit guides memory construction with a fixed Actor and framework. Our reset-based evaluation complements native runtimes, intermediate grading, and stateful harnesses (Ding et al., 2026; Li et al., 2026e; Ning et al., 2026).

Procedural Memory and Workflow Induction. Graph-based systems organize subtasks or symbolic abstractions (Zhang et al., 2026e; Ranaldi, 2026); G-Memory connects fine-grained interactions to queries and higher-level insights (Zhang et al., 2026a). Other methods extract workflows (Wang et al., 2024), maintain transferable routines (Cao et al., 2026b; Liang et al., 2026b; Sun et al., 2026), or distill reasoning memories and skill libraries (Ouyang et al., 2026; Xu et al., 2026). Retrieval aligns experience with current states or steps (Wang et al., 2026; Huang et al., 2025; Liu et al., 2025; Cui et al., 2025; Zhang et al., 2026c; Yang et al., 2025); MemoryAgentBench evaluates incremental memory (Hu et al., 2026). Traversal-as-Policy compiles logs into gated behavior trees (Li et al., 2026b); MemDocAgent uses traversal-guided repository memory (Bae et al., 2026). `Trace` uses delayed credit to identify candidates and backward slicing to link observed effects to target facts and estimated prerequisites, producing state-conditioned procedures for monitored execution.

Prerequisite-Aware Walkthrough Scheduling. Exploration uses posterior sampling or learned policies (Arumugam & Griffiths, 2026; Jiang et al., 2026), self-imitation or memory (Qin et al., 2026; Liu et al., 2026b), and world memory, offline imagination, or preference-guided tool use (Kim et al., 2026a; Hamidi et al., 2026; Li et al., 2026c). Skill-based agents reuse behaviors (Klissarov et al., 2025; Agashe et al., 2025; Tan et al., 2025); planning and orchestration select tools or subtasks (Yang et al., 2026; Nielsen et al., 2026; Zhao et al., 2025; Liang et al., 2026a). Our scheduler checks Walkthrough applicability against current facts and propagates value through producer–consumer relations, crediting preparation for downstream progress.

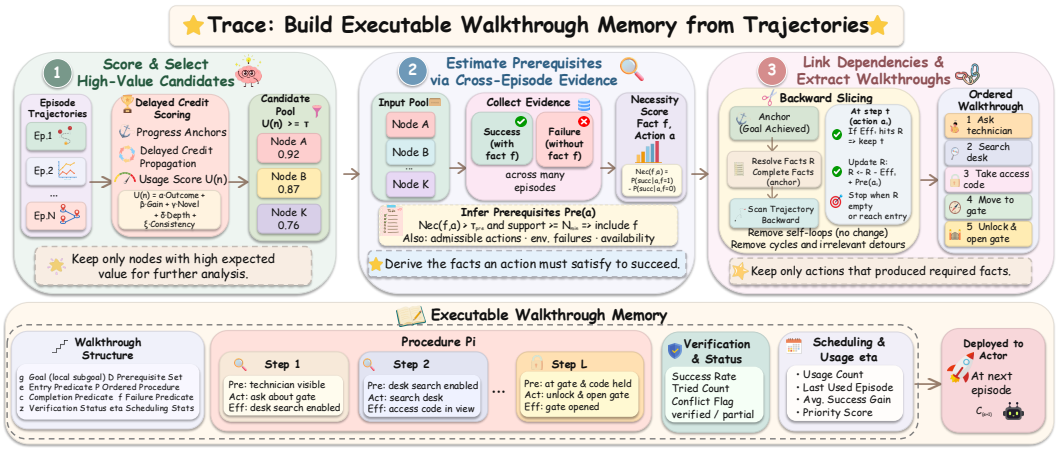


Figure 1: Overview of `Trace`. The Evolver scores progress, estimates candidate prerequisites across episodes, and slices trajectories backward from anchor facts to compile executable Walkthroughs. The scheduler checks their state conditions and guides the frozen Actor while monitoring effects and termination.

3 Credit-Guided Walkthrough Memory Construction and Scheduling

Problem and execution interface. `Trace` induces state-conditioned procedures from noisy, sparse-reward trajectories. A frozen LLM Actor repeatedly attempts the same task; each episode resets the environment and ends with a deterministic Evolver updating persistent *Walkthrough Memory*. Each Walkthrough is

$$W_i = (g_i, D_i, e_i, \Pi_i, c_i, f_i, z_i, \eta_i), \tag{1}$$

with goal g_i, candidate prerequisite facts D_i, entry predicate e_i, procedure Π_i, completion/failure predicates c_i, f_i, verification status z_i, and scheduling statistics η_i. Its L_i steps, $\Pi_i = (p_{i,1}, \ldots, p_{i,L_i})$, specify preconditions, actions, and expected effects: $p_{i,j} = (P_{i,j}, a_{i,j}, E_{i,j})$. The Actor generates primitive actions under this guidance.

Construction and reuse. Figure 1 connects progress anchoring and credit (Section 3.1), prerequisite estimation (Section 3.2), and backward slicing (Section 3.3): anchors delimit goals, credit identifies candidate preparation, and prerequisites link retained producers. The scheduler checks the resulting Walkthroughs against current facts and monitors execution (Section 3.4).

3.1 Progress Anchoring and Delayed Credit

Representing observable progress. The trajectory $\tau = \{(o_t, a_t, r_t, \mathrm{inv}_t)\}_{t=1}^{T}$ records observations, actions, cumulative rewards, and inventories over T actions. The Evolver parses comparable states $s_t = (\ell_t, \mathcal{I}_t, \mathcal{O}_t, \mathcal{Z}_t, \mathcal{E}_t, \mathcal{A}_t)$: location, inventory, visible objects, object states, exits, and admissible actions, yielding facts $\textsc{at}(\ell)$, $\textsc{has}(x)$, $\textsc{visible}(x)$, $\textsc{state}(x, v)$, $\textsc{exit}(d, \ell')$, and $\textsc{admissible}(a)$. Fixed parsers, regular expressions, and object lexicons recover missing structured fields. Lowercasing, alias normalization, deduplication, sorting, and fixed-order serialization yield canonical keys $\kappa_t = \mathrm{Hash}(s_t)$. Parser errors affect these facts; the Actor receives the original text.

For transition $e_t : \kappa_t \xrightarrow{a_t} \kappa_{t+1}$, changes $\Delta r_t = r_{t+1} - r_t$ and $\Delta s_t = \mathrm{Diff}(s_t, s_{t+1})$ define boundaries: *hard progress anchors* mark changes in reward, inventory, persistent object state, unlocked regions/exits, persistent flags, or terminal status; *discovery anchors* mark first visits, new objects/exits, newly admissible actions, or untried opportunities. Fixed templates label goals, e.g., "obtain access code."

Crediting preparatory actions. To credit preparation preceding an anchor, the Evolver computes local progress

$$g_t = \Delta r_t + \lambda_I I_t + \lambda_O O_t + \lambda_E E_t - \lambda_F F_t, \tag{2}$$

where deterministic templates map $\Delta r_t, \Delta s_t$ to task-relevant inventory (I_t), object-state (O_t), and

exploration (E_t) progress; F_t flags invalid actions, terminal failure, loops, or no progress. For a candidate segment ending at anchor step t_a, credit propagates backward:

$$C_{t_a} = g_{t_a}, \qquad C_t = g_t + \gamma C_{t+1}, \quad t < t_a, \tag{3}$$

where $\gamma \in (0, 1)$ discounts credit. All nonnegative progress weights and the discount are fixed across benchmarks (Appendix B.2). Credit identifies candidate preparation; action outcomes supply evidence about its prerequisites.

3.2 Estimating Candidate Prerequisites from Experience

To identify candidate prerequisites beyond temporal credit, the Evolver compares outcomes with and without each pre-action fact. For attempt i of action a, $x_i^f \in \{0, 1\}$ indicates pre-action observation of f, excluding effects of a. Success ($y_i = 1$) requires environment acceptance, no terminal failure or explicit invalid-action message, and an expected state change, newly admissible action, or progress toward an anchor; otherwise $y_i = 0$.

Across all observed attempts $\mathcal{D}_a$, support is $N_b(f, a) = \sum_{i \in \mathcal{D}_a} \mathbb{I}[x_i^f = b]$ for fact-present ($b = 1$) and fact-absent ($b = 0$) groups, with $\mathbb{I}[\cdot]$ the indicator function. Additive smoothing yields

$$\widehat{\Pr}(y = 1 \mid a, x^f = b) = \frac{\sum_{i \in \mathcal{D}_a} \mathbb{I}[x_i^f = b]\mathbb{I}[y_i = 1] \; + \; \alpha}{N_b(f, a) + 2\alpha}, \qquad b \in \{0, 1\}, \tag{4}$$

where $\alpha > 0$ smooths sparse evidence. The prerequisite score is the success gap:

$$\mathrm{Nec}(f, a) = \widehat{\Pr}(y = 1 \mid a, x^f = 1) - \widehat{\Pr}(y = 1 \mid a, x^f = 0). \tag{5}$$

Candidate prerequisites must exceed gap threshold τ_{pre} with support $n_{\min}$ in both groups:

$$f \in \widehat{\mathrm{Pre}}(a) \iff \mathrm{Nec}(f, a) > \tau_{\mathrm{pre}} \; \wedge \; N_1(f, a) \geq n_{\min} \; \wedge \; N_0(f, a) \geq n_{\min}. \tag{6}$$

Pairs lacking support remain unspecified; admissibility changes and environment failure reasons provide additional evidence. These observational associations guide slicing but do not certify necessity: correlated facts, unobserved attempts, and sparse failures can produce false positives or negatives.

3.3 Dependency Slicing and Walkthrough Compilation

Slicing connects anchor completion facts and candidate prerequisites to their observed producers. Let $\mathrm{Eff}_t = \mathrm{Diff}^+(s_t, s_{t+1})$ denote positive effects and $\mathrm{Diff}^-(s_t, s_{t+1})$ deleted or consumed facts. A transition with $\kappa_t = \kappa_{t+1}$ is a no-progress self-loop. Facts match ($f \simeq f'$) when their canonical predicates and arguments agree.

Tracing observed producers. Starting at t_a with unresolved facts $R \leftarrow \mathrm{CompleteFacts(anchor)}$, the backward scan retains the nearest producer of each requirement whose effect remains present at its downstream use. Deletion or consumption invalidates earlier support. Each retained transition replaces the requirements it justifies with its own candidate prerequisites:

$$R \leftarrow (R \setminus \{r \in R : \exists f \in \mathrm{Eff}_t, \; f \simeq r\}) \cup \widehat{\mathrm{Pre}}(a_t). \tag{7}$$

Algorithm 1 tracks active facts A and matched requirements J_t, returning the chronological retained chain Π.

The anchor transition is retained if it produces a completion fact. Reward- or terminal-only anchors use associated persistent changes; without these, they delimit procedures but add no fact-based prerequisite edges. Under sparse evidence, pruning removes only exact self-loops and explicit back-and-forth cycles. Missing prerequisites can cause required actions to be dropped: slicing guarantees neither global minimality nor dependency completeness.

Compiling the execution interface. The retained chain populates Π_i in Eq. 1, with entry and prerequisite conditions. The anchor defines completion; terminal failure, irreversible loss of a required fact, or an unrecoverable prerequisite state defines failure.

3.4 State-Conditioned Scheduling and Execution

Checking applicability after episode resets. A Walkthrough edge $W_i \rightarrow W_j$ matches W_i's completion/expected-effect fact to W_j's requirement under $\simeq$. Cycle-forming edges are rejected to

Algorithm 1: Backward dependency slicing from an anchor

Require: Trajectory τ, anchor step t_a, completion facts $\mathrm{CompleteFacts}(\mathrm{anchor})$, candidate prerequisites $\widehat{\mathrm{Pre}}$

$R \leftarrow \mathrm{CompleteFacts}(\mathrm{anchor})$; $\Pi \leftarrow [\,]$; $A \leftarrow$ facts active at $t_a + 1$
for $t = t_a, t_a - 1, \ldots, 1$ **do**
  $J_t \leftarrow \{r \in R : \exists f \in \mathrm{Eff}_t,\ f \simeq r,\ f \in A\}$
  if $J_t \neq \varnothing$ **then**
    prepend transition t to Π
    $R \leftarrow (R \setminus J_t) \cup \widehat{\mathrm{Pre}}(a_t)$
    record edges from retained producers of facts in $\widehat{\mathrm{Pre}}(a_t)$ to transition t when such producers are found
  end if
  $A \leftarrow (A \setminus \mathrm{Diff}^-(s_{t-1}, s_t)) \cup \mathrm{Diff}^+(s_{t-1}, s_t)$ for $t > 1$
end for
return chronological retained chain Π

keep G_W a directed acyclic graph (DAG); their facts may remain requirements but the edges cannot propagate value.

Historical dependencies do not ensure current applicability. Let $F(s_t)$ be current canonical facts, S_t, B_t the episode-local completed and blocked Walkthroughs, and $\mathrm{CompleteFacts}(S_t)$ the facts produced by completed Walkthroughs. Eligibility requires

$$\mathcal{C}_t = \{W_i \mid e_i(s_t) = 1,\ D_i \subseteq F(s_t) \cup \mathrm{CompleteFacts}(S_t),\ W_i \notin S_t \cup B_t\}\,. \tag{8}$$

This permits intermediate-state reuse when requirements hold. Deleted or consumed facts no longer support eligibility, even if previously produced in this episode (Appendix A.4).

Valuing preparation and selecting a procedure. Preparation is valued through local and downstream progress. Statistics $\eta_i = (n_i, \bar{Q}_i, M_{2,i}, V_i)$ record execution count, mean return, second-moment accumulator, and variance $V_i = M_{2,i}/\max(1, n_i - 1)$, counting completed and blocked executions. The return over executed primitive steps $\mathrm{exec}(W_i)$ is

$$Q_i = \sum_{t \in \mathrm{exec}(W_i)} g_t + \sum_{W_j \in \mathrm{Desc}(W_i)} \gamma_W^{d(i,j)}\, \widehat{Q}_j, \tag{9}$$

where γ_W discounts downstream value by shortest directed-path distance $d(i, j)$. The set $\mathrm{Desc}(W_i)$ counts each reachable descendant once; $\widehat{Q}_j = \bar{Q}_j$ for visited descendants and zero otherwise. Unvisited Walkthroughs use their prior variance only for their own selection. After completion or blocking, the scheduler updates

$$n_i \leftarrow n_i + 1, \quad \delta = Q_i - \bar{Q}_i, \quad \bar{Q}_i \leftarrow \bar{Q}_i + \delta/n_i, \quad M_{2,i} \leftarrow M_{2,i} + \delta(Q_i - \bar{Q}_i). \tag{10}$$

Unvisited eligible Walkthroughs are tried first. Otherwise, selection uses a Thompson-style score

$$u_i = \bar{Q}_i + \epsilon_i \sqrt{V_i/\max(1, n_i)}, \qquad \epsilon_i \sim \mathcal{N}(0, 1), \tag{11}$$

choosing $\arg\max_{W_i \in \mathcal{C}_t} u_i$ with ties broken by stored Walkthrough order.

Monitoring Actor execution. The prompt supplies the goal, entry/prerequisite conditions, current step, recommended action, expected effect, and termination predicates. After each action, the scheduler updates facts and checks effects and termination: it advances, retries, resumes at the nearest later step with a satisfied precondition, marks completion, or blocks after failure, repeated cycles, or effect failures. Appendices A.5 and B.2 give prompts, recovery rules, and scheduling settings.

4 Experiments

In this section, we conduct extensive experiments to answer the following research questions: **(RQ1)**: How does `Trace` compare with strong test-time learning and memory baselines? **(RQ2)**: Does each construction mechanism contribute to the intended diagnostic outcome? **(RQ3)** How do Walkthrough representation and dependency-aware scheduling affect cross-episode experience reuse? **(RQ4)**: How robust are Walkthrough construction and execution to controlled offline and online perturbations, and under what conditions does the method's advantage diminish?

Table 1: (RQ1-A) Overall performance comparison across three benchmarks. All methods are evaluated with the same backbone model within each block. We highlight the best and second-best results. †: No-learning Baseline. ⋆: Memory-based & Reflection-based Methods. ◇: Automated Prompt Optimization Methods.

Backbone	Method	WebShop		J-TTL		ScienceWorld		Avg.	
		AUC ↑	Final-3 ↑	AUC ↑	Final-3 ↑	AUC ↑	Final-3 ↑	AUC ↑	Final-3 ↑
Qwen3-32B	Static†	0.4823 ↑0.00	0.4706 ↑0.00	0.0290 ↑0.00	0.0191 ↑0.00	0.1397 ↑0.00	0.1150 ↑0.00	0.2170 ↑0.00	0.2016 ↑0.00
	Memory⋆	0.5403 ↑0.06	0.5578 ↑0.09	0.1327 ↑0.10	0.0583 ↑0.04	-0.0273 ↓0.17	-0.0020 ↓0.12	0.2152 ↓0.00	0.2047 ↑0.00
	RAG⋆	0.4510 ↓0.03	0.4321 ↓0.04	0.0263 ↓0.00	0.0180 ↓0.00	0.1200 ↓0.02	0.1170 ↑0.00	0.1991 ↓0.02	0.1890 ↓0.01
	Summary⋆	0.4680 ↓0.01	0.4943 ↑0.02	0.1461 ↑0.12	0.0654 ↑0.05	0.0759 ↓0.06	0.0624 ↓0.05	0.2300 ↑0.01	0.2074 ↑0.01
	Reflexion⋆	0.4859 ↑0.00	0.4728 ↑0.00	0.2380 ↑0.21	0.1870 ↑0.17	0.1388 ↓0.00	0.1513 ↑0.04	0.2876 ↑0.07	0.2704 ↑0.07
	EvoTest◇	0.4889 ↑0.01	0.5094 ↑0.04	0.1562 ↑0.13	0.0718 ↑0.05	0.1507 ↑0.01	0.1446 ↑0.03	0.2653 ↑0.05	0.2419 ↑0.04
	ACE◇	0.2128 ↓0.27	0.2317 ↓0.24	0.1855 ↑0.16	0.0768 ↑0.06	0.1530 ↑0.01	0.1602 ↑0.05	0.1838 ↓0.03	0.1563 ↓0.05
	APEX◇	0.5034 ↑0.02	0.5412 ↑0.07	0.1668 ↑0.14	0.1870 ↑0.17	0.0336 ↓0.11	0.0028 ↓0.11	0.2346 ↑0.02	0.2437 ↑0.04
	Trace	0.5421 ↑0.06	0.6047 ↑0.13	0.2410 ↑0.21	0.2833 ↑0.26	0.1884 ↑0.05	0.2186 ↑0.10	0.3238 ↑0.11	0.3689 ↑0.17
Mistral-Small-3.2-24B	Static†	0.3974 ↑0.00	0.3867 ↑0.00	0.0398 ↑0.00	0.0309 ↑0.00	0.1626 ↑0.00	0.1333 ↑0.00	0.1999 ↑0.00	0.1836 ↑0.00
	Memory⋆	0.4512 ↑0.05	0.5549 ↑0.17	0.1212 ↑0.08	0.0847 ↑0.05	0.1479 ↓0.01	0.1378 ↑0.00	0.2401 ↑0.04	0.2591 ↑0.08
	RAG⋆	0.3386 ↓0.06	0.3524 ↓0.03	0.0544 ↑0.01	0.0740 ↑0.04	0.1215 ↓0.04	0.1511 ↑0.02	0.1715 ↓0.03	0.1925 ↑0.01
	Summary⋆	0.4149 ↑0.02	0.5118 ↑0.13	0.1610 ↑0.12	0.1053 ↑0.07	0.1658 ↑0.00	0.1767 ↑0.04	0.2472 ↑0.05	0.2646 ↑0.08
	Reflexion⋆	0.4461 ↑0.05	0.4295 ↑0.04	0.2078 ↑0.17	0.1624 ↑0.13	0.1628 ↑0.00	0.2396 ↑0.11	0.2722 ↑0.07	0.2772 ↑0.09
	EvoTest◇	0.4627 ↑0.07	0.5268 ↑0.14	0.2073 ↑0.17	0.1608 ↑0.13	0.1630 ↑0.00	0.1914 ↑0.06	0.2777 ↑0.08	0.2930 ↑0.11
	ACE◇	0.2875 ↓0.11	0.2913 ↓0.10	0.1847 ↑0.14	0.1470 ↑0.12	0.1889 ↑0.03	0.2188 ↑0.09	0.2204 ↑0.02	0.2190 ↑0.04
	APEX◇	0.4318 ↑0.03	0.5682 ↑0.18	0.2250 ↑0.19	0.1808 ↑0.15	0.1927 ↑0.03	0.2055 ↑0.07	0.2832 ↑0.08	0.3182 ↑0.13
	Trace	0.4896 ↑0.09	0.5726 ↑0.19	0.3742 ↑0.33	0.3195 ↑0.29	0.2243 ↑0.06	0.2476 ↑0.11	0.3627 ↑0.16	0.3799 ↑0.20
GPT-OSS-20B	Static†	0.4523 ↑0.00	0.4189 ↑0.00	0.0253 ↑0.00	0.0130 ↑0.00	0.1731 ↑0.00	0.1949 ↑0.00	0.2169 ↑0.00	0.2089 ↑0.00
	Memory⋆	0.5123 ↑0.06	0.5567 ↑0.14	0.0567 ↑0.03	0.0548 ↑0.04	0.1530 ↓0.02	0.1661 ↓0.03	0.2407 ↑0.02	0.2592 ↑0.05
	RAG⋆	0.4210 ↓0.03	0.3812 ↓0.04	0.0076 ↓0.02	0.0220 ↑0.01	0.1058 ↓0.07	0.0996 ↓0.10	0.1781 ↓0.04	0.1676 ↓0.04
	Summary⋆	0.4380 ↓0.01	0.4915 ↑0.07	0.1155 ↑0.09	0.0622 ↑0.05	0.1497 ↓0.02	0.1486 ↓0.05	0.2344 ↑0.02	0.2341 ↑0.03
	Reflexion⋆	0.4529 ↑0.00	0.4263 ↑0.01	0.1105 ↑0.09	0.0687 ↑0.06	0.1732 ↑0.00	0.1877 ↓0.01	0.2455 ↑0.03	0.2276 ↑0.02
	EvoTest◇	0.4529 ↑0.00	0.5778 ↑0.16	0.1375 ↑0.11	0.0971 ↑0.08	0.1287 ↓0.04	0.1544 ↓0.04	0.2397 ↑0.02	0.2764 ↑0.07
	ACE◇	0.1728 ↓0.28	0.1632 ↓0.26	0.0913 ↑0.07	0.0413 ↑0.03	0.1401 ↓0.03	0.1394 ↓0.06	0.1347 ↓0.08	0.1146 ↓0.09
	APEX◇	0.4654 ↑0.01	0.5441 ↑0.13	0.0906 ↑0.07	0.0613 ↑0.05	0.1972 ↑0.02	-0.0833 ↓0.28	0.2511 ↑0.03	0.1740 ↓0.03
	Trace	0.5143 ↑0.06	0.5960 ↑0.18	0.3783 ↑0.35	0.4927 ↑0.48	0.2527 ↑0.08	0.3113 ↑0.12	0.3818 ↑0.16	0.4667 ↑0.26

4.1 Experimental Setup

Datasets and Benchmarks. To evaluate `Trace` across different forms of long-horizon interaction, we adopt three widely used benchmarks across three domains: **(1) Interactive Fiction**, including J-TTL (Hausknecht et al., 2020; He et al., 2026); **(2) E-commerce**, including WebShop (Yao et al., 2022); and **(3) Scientific Reasoning**, including ScienceWorld (Wang et al., 2022). J-TTL emphasizes prerequisite-heavy text-game execution, WebShop requires multi-step product search and selection, and ScienceWorld contains structured scientific tasks with object and state changes. Further implementation and accounting details appear in the Appendix.

Baselines. We evaluate eight baselines: (1) `Static`; (2) `Memory` (He et al., 2026); (3) `RAG` (He et al., 2026); (4) `Summary` (He et al., 2026); (5) `Reflexion` (Shinn et al., 2023); (6) `EvoTest` (He et al., 2026); (7) `ACE` (Zhang et al., 2026b); and (8) `APEX` (Li et al., 2026d).

LLM Backbones. For instantiating these frameworks, we adopt three open-source LLMs, `Qwen3-32B`, `Mistral-Small-3.2-24B`, and `GPT-OSS-20B`.

Metrics. We report **Final-**3, the mean normalized score over the last three episodes, and **AUC**, the mean over all episodes following He et al. (2026). **Replay Success (RS)** measures the fraction of Walkthrough attempts completed successfully; **Prerequisite Violation (PV)** measures the fraction of attempted steps inconsistent with current-state preconditions or inferred prerequisites. PV diagnoses execution consistency, not ground-truth prerequisite accuracy. **Token Cost** counts all input and output tokens. Scores are normalized without clipping, retaining benchmark-native penalties. Full definitions, formulas, and benchmark-specific normalization appear in Appendix B.1.

Parameter Configurations. For all experiments, we set the LLM temperature to 0.8. Each task is run for 50 episodes, with each episode limited to at most 150 environment steps. Within each benchmark-backbone block, all compared methods use the same episode budget, primitive environment-step budget, backbone model, task ordering, and benchmark scoring function. The

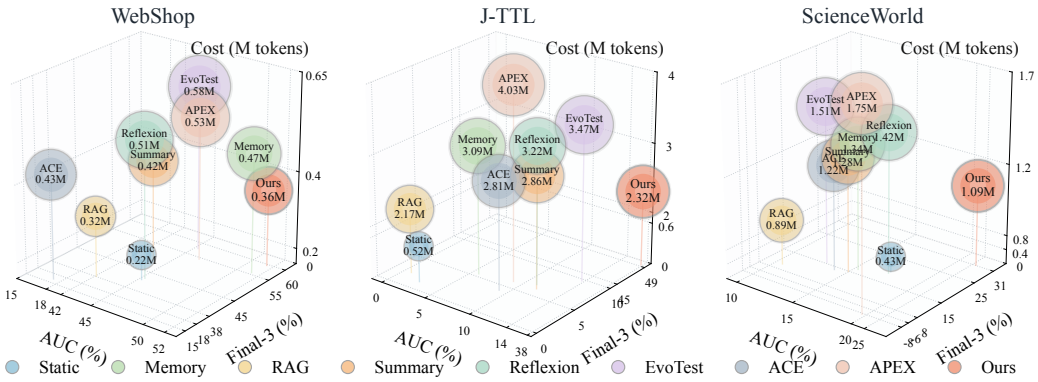


Figure 2: (RQ1-B) Performance–cost comparison using `GPT-OSS-20B`. All methods use the same backbone and are evaluated under the same 50-episode interaction budget. Total Tokens include both input and output tokens consumed over all episodes and are reported in millions (M). We highlight the best and second-best performance.

construction, prerequisite-estimation, scheduling, and baseline-prompt hyperparameters are reported in Appendix B.2. The reported tables use point estimates; supplemental perturbation results are averaged over the three paired seeds.

4.2 Main Results (RQ1)

Takeaway 1: `Trace` achieves the best performance across tasks and backbone models. As shown in Table 1, we compute the relative gain over the strongest baseline in the Avg. column separately for each backbone and metric, then take the arithmetic mean of these gains across the three backbones. Under this aggregation, **`Trace`** improves benchmark-averaged AUC and Final-3 by 30.9% and 41.6%, respectively. On `Qwen3-32B`, `Mistral-Small-3.2-24B`, and `GPT-OSS-20B`, the AUC gains are 12.6%, 28.1%, and 52.1%, while the Final-3 gains are 36.4%, 19.4%, and 68.8%, respectively. The improvements are most pronounced on J-TTL: applying the same calculation to its strongest baseline for each backbone and metric gives mean relative gains of 80.9% in AUC and 178.5% in Final-3.

Takeaway 2: `Trace` achieves its performance gains without incurring excessive inference cost. As shown in Figure 2, **`Trace`** consumes only 0.3564M, 2.3231M, and 1.0896M tokens on WebShop, J-TTL, and ScienceWorld, respectively. Token cost includes both input and output tokens from Actor calls and any LLM calls used by a method for reflection, summarization, retrieval-conditioned prompting, or prompt optimization. The **`Trace`** Evolver uses deterministic parsing, state differencing, dependency slicing, and scheduling, so these local operations do not add LLM-token cost. **`Trace`** is consistently lower than Memory, Summary, Reflexion, EvoTest, ACE, and APEX across all three benchmarks, and is also more efficient than the AUC-best baseline in each setting: it uses 23.6% fewer tokens than Memory on WebShop, 33.1% fewer than EvoTest on J-TTL, and 37.8% fewer than APEX on ScienceWorld. These results support the interpretation that the observed gains are not obtained by simply increasing inference-token expenditure.

4.3 Ablation Study (RQ2)

Takeaway 3: All three construction stages provide complementary and cumulative gains in end-to-end performance and execution diagnostics. As shown in Table 2, adding Delayed Credit to Immediate Progress Only improves AUC by 0.0381 and Final-3 by 0.0469 on ScienceWorld, while improving AUC by 0.0734 and Final-3 by 0.0975 on J-TTL. Prerequisite Estimation provides AUC and Final-3 gains

Table 2: (RQ2) Nested ablation of the three Walkthrough construction stages illustrated in Figure 1, evaluated on ScienceWorld and J-TTL using `GPT-OSS-20B`.

Del. Cred.	Prereq. Est.	Back. Slic.	ScienceWorld AUC ↑	ScienceWorld Final-3 ↑	J-TTL AUC ↑	J-TTL Final-3 ↑
✓	✓	✓	0.2527 ↑0.00	0.3113 ↑0.00	0.3783 ↑0.00	0.4927 ↑0.00
✓	✓	×	0.1841 ↓0.07	0.2284 ↓0.08	0.3116 ↓0.07	0.4072 ↓0.09
✓	×	×	0.1453 ↓0.04	0.1815 ↓0.05	0.2398 ↓0.07	0.3158 ↓0.09
×	×	×	0.1072 ↓0.04	0.1346 ↓0.05	0.1664 ↓0.07	0.2183 ↓0.10

of 0.0388 and 0.0469 on ScienceWorld,
and 0.0718 and 0.0914 on J-TTL. Backward Dependency Slicing further increases AUC by 0.0686 and Final-3 by 0.0829 on ScienceWorld, and increases AUC by 0.0667 and Final-3 by 0.0855 on J-TTL. Overall, the full method exceeds Immediate Progress Only by 0.1455 in AUC and 0.1767 in Final-3 on ScienceWorld, and by 0.2119 in AUC and 0.2744 in Final-3 on J-TTL. Because this is a nested ablation, it supports the cumulative value of the construction stages rather than isolating every possible causal interaction among them.

4.4 Case Study(RQ3)

Takeaway 4: Executable representation and dependency-aware scheduling improve execution efficiency and inferred-prerequisite consistency, respectively, and provide complementary benefits. As shown in Figure 3, enabling either mechanism alone improves every metric on both ScienceWorld and J-TTL. Executable representation primarily reduces execution cost, lowering Actions per Success from 29.4 to 17.8 on ScienceWorld and from 42.7 to 21.9 on J-TTL. Dependency-aware scheduling more strongly improves reuse validity: it increases Replay Success by 0.125 and 0.142, while reducing Prerequisite Violation by 0.145 and 0.212, on ScienceWorld and J-TTL, respectively. Each mechanism provides further gains when added on top of the other, and the full method achieves the best results across all metrics.

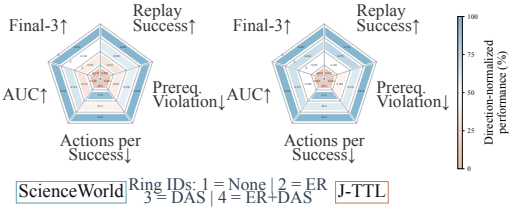


Figure 3: (RQ3) Effects of executable Walkthrough representation and dependency-aware scheduling on cross-episode experience reuse. All variants use the same nodes, action sequences, memory snapshots, and interaction budgets; the disabled variants remove only the corresponding execution representation or dependency-aware selection logic.

4.5 Construction and Online Robustness Study (RQ4)

Takeaway 5: Delayed Credit remains effective under partial masking of non-terminal progress signals, but relying only on terminal rewards substantially degrades Walkthrough executability and downstream performance. As shown in Figure 4, masking 25% of the non-terminal progress signals reduces Replay Success by only 0.054–0.061, while AUC and Final-3 decrease by 0.0333–0.0578 and 0.0430–0.0731, respectively. All three metrics decline consistently as more intermediate signals are removed. When only terminal rewards are retained, Replay Success decreases by 0.178–0.240, accompanied by substantially larger reductions in AUC and Final-3. These results show that Delayed Credit tolerates partially missing progress evidence, whereas intermediate state changes remain essential for constructing executable Walkthroughs and improving subsequent episodes.

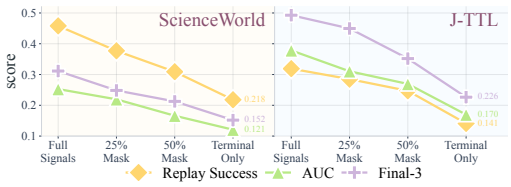


Figure 4: (RQ4-A) Robustness to progress-signal masking on ScienceWorld and J-TTL using `GPT-OSS-20B`.

Takeaway 6: Additional cross-episode evidence improves endpoint performance within the 50-episode interaction budget. As shown in Figure 5, increasing the available history from 3 to 9 episodes improves Replay Success, AUC, and Final-3 by 0.010, 0.036, and 0.110 on ScienceWorld, and by 0.107, 0.116, and 0.106 on J-TTL, respectively. The intermediate results are not strictly monotonic, but the nine-episode setting outperforms the three-episode setting on every metric and both benchmarks. These results show that Prerequisite Estimation can benefit from additional early success and failure observations within the practical 50-episode interaction budget.

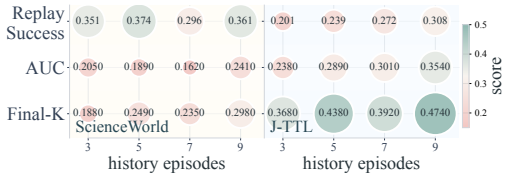


Figure 5: (RQ4-B) Cross-episode evidence sample efficiency on ScienceWorld and J-TTL using `GPT-OSS-20B`.

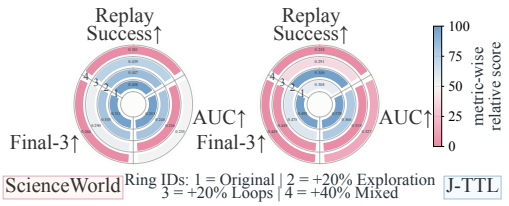


Figure 6: (RQ4-C) Robustness to trajectory noise on ScienceWorld and J-TTL using `GPT-OSS-20B`.

Takeaway 7: The Walkthrough construction pipeline is robust to controlled offline trajectory noise, retaining most of its clean-trajectory performance even under substantial structured redundancy. As shown in Figure 6, with 20% isolated exploratory actions inserted into successful construction trajectories, the metrics that decrease retain 96.3%–97.6% of their original performance across both benchmarks, while Replay Success on J-TTL increases from 0.308 to 0.346. We treat this increase as evidence that additional valid exploration can occasionally expose useful intermediate facts, rather than as a monotonic robustness guarantee. Under the more challenging setting with 20% loops and backtracking, the method preserves 84.4%–95.9% of its clean-trajectory performance. Even when mixed redundancy reaches 40%, it retains 83.2%–92.9% of the corresponding clean values. These results evaluate robustness of the Evolver's construction input, not full online robustness to adversarial or arbitrarily corrupted Actor behavior, and the method still depends on the availability and quality of observable state evidence.

Takeaway 8: The online loop recovers from most controlled perturbations, with the largest performance losses under forced Actor deviations. As shown in Table 3, we evaluate the full online loop under parser-field drops, object-alias substitutions, action-list corruption, and forced Actor deviations, each applied independently at a matched 20% event rate on ScienceWorld and J-TTL with `GPT-OSS-20B`. Each condition uses the same task split, 50-episode budget, 150-step limit, and three paired random seeds as the clean runs; Appendix B.4.2 details the perturbation operators and recovery criterion. Table entries are means over these seeds, with parentheses showing paired Final-3 changes from clean. Averaged across benchmarks and perturbation types, Final-3 decreases by 0.0398, while the scheduler recovers or resynchronizes without blocking in 72.0% of perturbed Walkthrough executions. Alias substitution yields the smallest performance decrease on both benchmarks, whereas forced Actor deviations yield the largest, reaching 0.0801 on J-TTL. Recovery ranges from 55.8% under Actor deviation on J-TTL to 86.7% under alias substitution on ScienceWorld. These results support recovery under controlled single-source test-time corruption, with greater sensitivity to deviations from recommended actions; compound errors and unconstrained failures remain outside the evaluated scope.

Table 3: (RQ4-D) Online robustness of the full test-time loop.

Benchmark	Online condition	Final-3 ↑	Recovery (%) ↑
ScienceWorld	Clean	0.3113	–
	Parser-field drop	0.2861 ($\Delta - 0.0252$)	78.4
	Alias substitution	0.2976 ($\Delta - 0.0137$)	86.7
	Action-list corruption	0.2748 ($\Delta - 0.0365$)	71.2
	Actor deviation	0.2595 ($\Delta - 0.0518$)	62.5
J-TTL	Clean	0.4927	–
	Parser-field drop	0.4562 ($\Delta - 0.0365$)	72.8
	Alias substitution	0.4729 ($\Delta - 0.0198$)	82.1
	Action-list corruption	0.4381 ($\Delta - 0.0546$)	66.4
	Actor deviation	0.4126 ($\Delta - 0.0801$)	55.8

5 Conclusion

We introduce **`Trace`** to learn executable Walkthroughs from noisy, sparse-reward trajectories for reliable experience reuse. **`Trace`** combines delayed-credit propagation, fact-grounded prerequisite estimation, and backward dependency slicing to extract action chains supported by observed state-dependency evidence. Empirically, its relative gains over the strongest baselines in benchmark-averaged AUC and Final-3 are 30.9% and 41.6%, respectively, averaged across backbones, with lower token cost. Ablations and controlled construction-noise studies support complementary gains from the construction stages and stability of the extraction pipeline to redundant trajectory steps under the tested perturbations. Controlled online perturbations on ScienceWorld and J-TTL with `GPT-OSS-20B` show moderate, perturbation-dependent degradation under matched single-source corruption. Taken together, the results characterize both the benefits and the observable failure modes of state-conditioned, monitorable procedures for efficient and resumable long-horizon agents under the evaluated benchmark protocols.

AI USE STATEMENT

Generative AI tools were used solely to polish the language of this manuscript. All research ideas, experiments, and analyses are the authors' own. The authors reviewed the AI-assisted edits and take full responsibility for the content of this paper.

ETHICS STATEMENT

This work evaluates agents in simulated environments and does not involve human participants or personal data. We are not aware of specific ethical risks beyond those associated with general-purpose LLM agents.

REPRODUCIBILITY STATEMENT

The method, experimental setup, baselines, and evaluation metrics are described in the main text. Additional implementation details and hyperparameters are provided in the appendix.

REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Wang. Agent s: An open agentic framework that uses computers like a human. In *International Conference on Learning Representations*, volume 2025, pp. 22924–22946, 2025.

Dilip Arumugam and Thomas L. Griffiths. Toward efficient exploration by large language model agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=M3vwnscpL2.

Suyoung Bae, Jaehoon Lee, Changkyu Choi, YunSeok Choi, and Jee-Hyong Lee. Remember your trace: Memory-guided long-horizon agentic framework for consistent and hierarchical repository-level code documentation. *arXiv preprint arXiv:2605.14563*, 2026. URL https://arxiv.org/abs/2605.14563.

Nikolas Belle, Dakota Barnes, Alfonso Amayuelas, Ivan Bercovich, Xin Eric Wang, and William Wang. Agents of change: Self-evolving LLM agents for strategic planning. *arXiv preprint arXiv:2506.04651*, 2025. URL https://arxiv.org/abs/2506.04651.

Yishuo Cai, Xingyu Guo, Xuancheng Huang, Jinhua Du, Can Huang, Wenxuan Huang, Wenhan Ma, Yuyang Hu, Aohan Zeng, Jie Tang, and Xu Sun. From player to master: Enhancing test-time learning of LLM agents via reinforcement learning over memory. *arXiv preprint arXiv:2606.08656*, 2026. URL https://arxiv.org/abs/2606.08656.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16803–16822, 2026a.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16803–16822, San Diego, California, United States, July 2026b. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.829. URL https://aclanthology.org/2026.findings-acl.829/.

Sijia Cui, Aiyao He, Shuai Xu, Hongming Zhang, Yanna Wang, Qingyang Zhang, Yajing Wang, and Bo Xu. Self-guided function calling in large language models via stepwise experience recall. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 10842–10854. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.574.

Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Yang JingYi, Penghui Yang, Zhixiong Zhang, Xilin Wei, Xinyu Fang, et al. Wildclawbench: A benchmark for real-world, long-horizon agent evaluation. *arXiv preprint arXiv:2605.10912*, 2026. URL https://arxiv.org/abs/2605.10912.

Tianqing Fang, Hongming Zhang, Zhisong Zhang, Kaixin Ma, Wenhao Yu, Haitao Mi, and Dong Yu. Webevolver: Enhancing web agent self-improvement with co-evolving world model. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 8970–8986, 2025.

Yubin Ge, Salvatore Romeo, Jason Cai, Monica Sunkara, and Yi Zhang. Samule: Self-learning agents enhanced by multi-level reflection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 16602–16621, 2025.

Shayan Mohajer Hamidi, Linfeng Ye, and Konstantinos N Plataniotis. Dreamphase: Offline imagination and uncertainty-guided planning for large-language-model agents. In *The Fourteenth International Conference on Learning Representations*, 2026.

Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7903–7910, 2020.

Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. Evotest: Evolutionary test-time learning for self-improving agentic systems. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=JFnnajbkvP.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2507.05257.

Tenghao Huang, Kinjal Basu, Ibrahim Abdelaziz, Pavan Kapanipathi, Jonathan May, and Muhao Chen. R2d2: Remembering, replaying and dynamic decision making with a reflective agentic memory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30318–30330, 2025.

Yulun Jiang, Liangze Jiang, Damien Teney, Michael Moor, and Maria Brbic. Meta-RL induces exploration in language agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=4GiBscHW1k.

Minsoo Kim et al. Dual-scale world memory for llm agents towards hard-exploration problems. In *The Fourteenth International Conference on Learning Representations*, 2026a.

Sunghwan Kim, Junhee Cho, Beong-woo Kwak, Taeyoon Kwon, Liang Wang, Nan Yang, Xingxing Zhang, Furu Wei, and Jinyoung Yeo. On training large language models for long-horizon tasks: An empirical study of horizon length. *arXiv preprint arXiv:2605.02572*, 2026b. URL https://arxiv.org/abs/2605.02572.

Martin Klissarov, Mikael Henaff, Roberta Raileanu, Shagun Sodhani, Pascal Vincent, Amy Zhang, Pierre-Luc Bacon, Doina Precup, Marlos C Machado, and Pierluca D'Oro. Maestromotif: Skill design from artificial intelligence feedback. In *International Conference on Learning Representations*, volume 2025, pp. 18029–18057, 2025.

Jiazheng Li, Yawei Wang, Qiaojing Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. SALT: Step-level advantage assignment for long-horizon agents via trajectory graph. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 4709–4725. Association for Computational Linguistics, 2026a. doi: 10.18653/v1/2026.findings-eacl.247. URL https://aclanthology.org/2026.findings-eacl.247/.

Peiran Li, Jiashuo Sun, Fangzhou Lin, Shuo Xing, Tianfu Fu, Suofei Feng, Chaoqun Ni, and Zhengzhong Tu. Traversal-as-policy: Log-distilled gated behavior trees as externalized, verifiable policies for safe, robust, and efficient agents. *arXiv preprint arXiv:2603.05517*, 2026b. URL https://arxiv.org/abs/2603.05517.

Pengxiang Li, Zhi Gao, Bofei Zhang, Yapeng Mi, Xiaojian Shawn Ma, Chenrui Shi, Tao Yuan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, et al. Iterative tool usage exploration for multimodal agents via step-wise preference tuning. *Advances in Neural Information Processing Systems*, 38: 59496–59528, 2026c.

Yibo Li, Jiashuo Yang, Zhi Zheng, Zhiyuan Hu, Yuan Sui, Shizun Wang, Yufei He, and Bryan Hooi. Apex: Autonomous policy exploration for self-evolving llm agents. *arXiv preprint arXiv:2605.21240*, 2026d.

Zongxia Li, Zhongzhi Li, Yucheng Shi, Ruhan Wang, Junyao Yang, Zhichao Liu, Xiyang Wu, Anhao Li, Yue Yu, Ninghao Liu, Lichao Sun, Haotao Mi, and Leowei Liang. Long-horizon-terminal-bench: Testing the limits of agents on long-horizon terminal tasks with dense reward-based grading. *arXiv preprint arXiv:2607.08964*, 2026e. URL https://arxiv.org/abs/2607.08964.

Dingkang Liang, Cheng Zhang, Xiaopeng Xu, Jianzhong Ju, Zhenbo Luo, and Xiang Bai. Cook and clean together: Teaching embodied agents for parallel task execution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 18415–18424, 2026a.

Sirui Liang, Pengfei Cao, Jian Zhao, Wenhao Teng, Xiangwen Liao, Jun Zhao, and Kang Liu. Learning how to remember: A meta-cognitive management method for structured and transferable agent memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 30733–30753, San Diego, California, United States, July 2026b. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1535. URL https://aclanthology.org/2026.findings-acl.1535/.

Sirui Liang, Pengfei Cao, Jian Zhao, Wenhao Teng, Xiangwen Liao, Jun Zhao, and Kang Liu. Learning how to remember: A meta-cognitive management method for structured and transferable agent memory. *arXiv preprint arXiv:2601.07470*, 2026c.

Minghao Liu, Yu Wang, Jiayun Wang, and Wei Wei. Reward-free evolving agents via pairwise validator. *arXiv preprint arXiv:2607.14408*, 2026a. URL https://arxiv.org/abs/2607.14408.

Yitao Liu, Chenglei Si, Karthik R Narasimhan, and Shunyu Yao. Contextual experience replay for self-improvement of language agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14179–14198, 2025.

Zeyuan Liu, Jeonghye Kim, Xufang Luo, Dongsheng Li, and Yuqing Yang. Exploratory memory-augmented LLM agent via hybrid on- and off-policy optimization. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=UOzxviKVFO.

Stefan Nielsen, Edoardo Cetin, Peter Schwendeman, Qi Sun, Jinglue Xu, and Yujin Tang. Learning to orchestrate agents in natural language with the conductor. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=U23A2BUKYt.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026. URL https://arxiv.org/abs/2605.18747.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=jL7fwchScm.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 79791–79821, 2025.

Yulei Qin, Xiaoyu Tan, Zhengbao He, Gang Li, Haojia Lin, Zongyi Li, Zihan Xu, Yuchen Shi, Siqi Cai, Renting Rui, Shaofei Cai, Yuzheng Cai, Xuan Zhang, Sheng Ye, Ke Li, and Xing Sun. Learn the ropes, then trust the wins: Self-imitation with progressive exploration for agentic reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=Kssko33Ekq.

Leonardo Ranaldi. Evolving agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16561–16569. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.754.

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.

Libo Sun, Jiwen Zhang, Siyuan Wang, and Zhongyu Wei. MAGNET: Towards adaptive GUI agents with memory-driven knowledge evolution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28181–28206. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1299.

Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, Ruyi An, Molei Qin, Chuqiao Zong, Longtao Zheng, Yujie Wu, Xiaoqiang Chai, Yifei Bi, Tianbao Xie, Pengjie Gu, Xiyun Li, Ceyao Zhang, Long Tian, Chaojie Wang, Xinrun Wang, Börje F. Karlsson, Bo An, Shuicheng Yan, and Zongqing Lu. Cradle: Empowering foundation agents towards general computer control. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://proceedings.mlr.press/v267/tan25h.html.

Leitian Tao, Baolin Peng, Wenlin Yao, Tao Ge, Hao Cheng, Mike Hang Wang, Jianfeng Gao, and Sharon Li. TRACE: Turn-level reward assignment via credit estimation for long-horizon agents. *arXiv preprint arXiv:2607.13988*, 2026. URL https://arxiv.org/abs/2607.13988.

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.

Tong Wang, Pei Xu, Shiyue Cao, Likun Yang, Daipeng Li, Jianbin Jiao, and Kaiqi Huang. Samem: State-aware memory as a fine-grained memory for llm agents in decision-making. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 14691–14710, 2026.

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

Han Xiao, Guozhi Wang, Yuxiang Chai, Zimu Lu, Weifeng Lin, Hao He, Lue Fan, Liuyang Bian, Rui Hu, Liang Liu, et al. Ui-genie: A self-improving approach for iteratively boosting mllm-based mobile gui agents. *Advances in Neural Information Processing Systems*, 38:150376–150411, 2026.

Weijia Xu, Alessandro Sordoni, Chandan Singh, Zelalem Gero, Michel Galley, Xingdi Yuan, and Jianfeng Gao. Test-time learning with an evolving library. *arXiv preprint arXiv:2605.14477*, 2026. URL https://arxiv.org/abs/2605.14477.

Shuo Yang, Caren Han, Yihao Ding, Shuhe Wang, and Eduard Hovy. Tooltree: Efficient llm tool planning via dual-feedback monte carlo tree search and bidirectional pruning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Wei Yang, Jinwei Xiao, Hongming Zhang, Qingyang Zhang, Yanna Wang, and Bo Xu. Coarse-to-fine grounded memory for LLM agent planning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13029–13056, 2025.

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.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, 2025.

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

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025a.

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. *arXiv preprint arXiv:2510.04618*, 2025b.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026b.

Wenyuan Zhang, Xinghua Zhang, Haiyang Yu, Shuaiyi Nie, Bingli Wu, Juwei Yue, Tingwen Liu, and Yongbin Li. ExpSeek: Self-triggered experience seeking for web agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 29255–29283. Association for Computational Linguistics, 2026c. doi: 10.18653/v1/2026.findings-acl.1462.

Yudi Zhang, Meng Fang, Zhenfang Chen, and Mykola Pechenizkiy. Self-evolving LLM agents with in-distribution optimization. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *PMLR*, 2026d. URL https://arxiv.org/abs/2606.07367.

Zhenkui Zhang, Wendong Bu, Kaihang Pan, Bingchen Miao, Wenqiao Zhang, Guoming Wang, Wei Ji, Rui Tang, Juncheng Li, and Siliang Tang. Evolving generalist virtual agents with generative and associative memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 13006–13014, 2026e.

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. doi: 10.1609/aaai.v38i17.29936.

Di Zhao, Longhui Ma, Siwei Wang, Miao Wang, and Zhao Lv. Cola: Collaborative multi-agent framework with dynamic task scheduling for gui automation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 4570–4593, 2025.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Yifei Zhou, Sergey Levine, Jason Weston, Xian Li, and Sainbayar Sukhbaatar. Self-challenging language model agents. *Advances in Neural Information Processing Systems*, 38:113959–113991, 2026.

A Method and Implementation Details

A.1 State Facts and Action Outcomes

The Evolver converts each observation into typed facts over location, inventory, visible objects, object states, exits, admissible actions, and persistent flags. Facts are canonicalized by lowercasing, alias normalization, deduplication, sorting, and deterministic serialization. When structured fields are unavailable, fixed parsers, regular expressions, and benchmark-specific object lexicons recover the missing fields. These parser-derived facts are used only by the Evolver; the Actor still receives the original observation text.

For prerequisite estimation, an action attempt is successful when the environment accepts the action, no terminal failure or explicit invalid-action message is produced, and the transition yields an expected state change, newly admissible action, or progress anchor. Attempts without sufficient positive and negative support are left unspecified rather than forced into a prerequisite relation. The resulting prerequisite set is therefore a candidate relation estimated from observational evidence, not a causal certificate.

A.2 Backward Slicing and Dependency Construction

Backward slicing operates over positive state effects. If multiple retained transitions can produce the same unresolved fact, the scan chooses the nearest observed producer that remains valid at the downstream use point. Facts deleted or consumed by later transitions are removed from the active fact set. This rule supports compact observed-fact-consistent extraction, but it is still a heuristic over observed trajectories and does not prove global minimality in environments with aliases, consumable resources, reversible predicates, or several equivalent producer paths.

For an anchor at step t_a, let R denote unresolved facts that must be justified, A_u denote facts still active at a downstream use point u, and $\mathrm{Del}_t = \mathrm{Diff}^-(s_t, s_{t+1})$ denote facts removed or consumed by transition t. The anchor transition is included in the scan: if transition t_a produced one of the completion facts, it is retained as the final step of the Walkthrough. The downstream use point for fact f is the nearest already retained later transition whose precondition or completion predicate mentions f; if no such transition exists, it is the anchor step. Fact matching uses the canonical relation $f \simeq f'$ defined in Section 3.3. The backward scan uses the following operational rule:

1. Initialize R with the anchor completion facts and set the retained transition list to empty.
2. Scan $t = t_a, \ldots, 1$. Before testing transition t, remove from active downstream support any fact deleted by transitions between t and its downstream use point.
3. Retain transition t if some $f \in \mathrm{Eff}_t$ matches an unresolved fact in R under $\simeq$ and at least one produced fact remains active at its downstream use point.
4. When transition t is retained, remove its justified facts from R, add $\widehat{\mathrm{Pre}}(a_t)$ to R, and record a dependency edge from the nearest retained producer of each prerequisite fact to the retained transition that uses it.
5. If two retained transitions produce the same unresolved fact, choose the nearest producer that remains active at the downstream use point. Earlier producers are kept only when they justify a different active fact.
6. Match aliases only after canonicalization. If canonicalization cannot reconcile an alias, the fact is treated as different and the resulting relation remains unsupported.

Consumable resources and nonpersistent effects are handled through Del_t: once a later transition consumes or deletes a fact, an earlier producer cannot justify a downstream use past that deletion. This semantics is still limited to observed state differences and does not prove that a retained transition is causally necessary in the environment.

A.3 Walkthrough Graph and Scheduling

The prerequisite graph used by the scheduler is a memory-level graph, not an environment-state graph. Each edge points from a Walkthrough that historically produced a canonical fact to a Walkthrough whose entry predicate or required facts contain the same canonical fact under $\simeq$. Candidate edges are considered in deterministic insertion order after each Evolver update. An edge is inserted only if it does not create a directed cycle in the current graph. If a candidate edge would create a cycle, the matched fact remains part of the consumer's eligibility check, but the edge is omitted from G_W so that topological value computation remains well-defined.

For value propagation, the scheduler first topologically orders the current DAG. When computing Q_i, it enumerates the unique downstream Walkthroughs reachable from W_i and uses the shortest directed-path distance $d(i, j)$ as the discount exponent in Eq. 9. Thus a shared descendant in a diamond-shaped graph contributes once to Q_i, not once per path. Downstream Walkthroughs with no observed executions use $\widehat{Q}_j = 0$ in value propagation; their prior variance remains available only for their own Thompson-style selection score if they become eligible. The online update in Eq. 10 is applied after a Walkthrough either completes or is blocked, using the observed local progress and the current downstream-value snapshot.

A.4 Episode Reset and Walkthrough Eligibility

Benchmark environments reset according to their standard episode protocols. `Trace` does not assume that acquired objects, opened exits, or other environment facts persist across episodes. What persists is the external Walkthrough Memory: a record of how a fact was previously produced, when a procedure was applicable, and how it was completed or failed.

At execution time, eligibility is evaluated against the current canonical state. A Walkthrough may be selected if its entry predicate holds and each required fact is either currently true or has been produced by a completed Walkthrough in the same episode. If a later action consumes or deletes a required fact, the downstream Walkthrough is no longer eligible until the fact is re-established. This distinction prevents cross-episode procedural knowledge from being mistaken for persistent cross-episode environment state.

A.5 Actor Prompt Interface and Execution Recovery

`Trace` does not replace the primitive Actor with a symbolic controller. The scheduler injects the selected Walkthrough as structured guidance in the Actor prompt, while the LLM still emits primitive environment actions. The prompt interface contains the task instruction, the original current observation text, the current inventory when available, the selected Walkthrough goal, entry and prerequisite facts, the current step precondition, the recommended action, the expected effect, the completion predicate, and the failure predicate. The Actor is instructed to produce a single valid environment action; after the action, the scheduler checks the observed effect against the Walkthrough state predicates and either advances, retries the same step, skips to a satisfied later step, marks completion, or blocks the Walkthrough after repeated cycles or expected-effect failures.

> **Structured guidance fields:** `goal`; `current_observation`; `inventory`; `required_facts`; `current_step_precondition`; `recommended_action`; `expected_effect`; `completion_predicate`; `failure_predicate`.

SYSTEM: You are an expert long-horizon environment actor. The current observation is authoritative. Use the Walkthrough only when its entry condition and current-step condition are consistent with the observation; otherwise follow the observation and valid-action list. USER fields are given in this order: current observation, inventory, valid actions when exposed by the environment, selected Walkthrough goal, entry condition, required facts, current-step condition, recommended action, expected effect, completion condition, failure condition, and the last five actions and observations. Required output format is exactly two lines: REASONING: one concise sentence explaining why the action is valid; ACTION: one primitive environment action. If valid actions are supplied, ACTION must exactly match one listed action. Deviation policy: accept a non-recommended action only when it is valid and either satisfies the current-step condition or moves toward the expected effect; otherwise retry the same Walkthrough step for at most two attempts and then block the Walkthrough. Contradiction policy: when the observation contradicts the entry condition, current-step condition, or expected effect, pause replay, resynchronize to the nearest later Walkthrough step whose condition is already satisfied, and block if no such step exists. Invalid-action policy: retry with a valid-action reminder for at most two attempts, then block the Walkthrough and return control to normal exploration.

B Experimental Setup

B.1 Evaluation Metrics and Score Normalization

Task performance. Each task is evaluated over $E = 50$ episodes. Final-3 is the average normalized score over the last three episodes. Following He et al. (2026), AUC averages the normalized score over the full episode budget:

$$\mathrm{AUC} = \frac{1}{E}\sum_{e=1}^{E}\frac{R(e)}{R_{\max}},$$

where $R(e)$ is the score obtained in episode e and $R_{\max}$ is the maximum score achievable in a single episode.

Score normalization. We keep benchmark-native rewards before normalization. Therefore, if a benchmark assigns negative raw penalties, the corresponding normalized AUC or Final-3 entry can be below zero rather than clipped to $[0, 1]$. ScienceWorld uses the episode score returned by the task wrapper as $R(e)$, sets $R_{\max} = 1.0$ after task-level score normalization, and reports unclipped AUC and Final-3. Negative entries are treated as wrapper-preserved penalty scores rather than post-hoc clipped official scores.

Replay Success (RS). RS is the fraction of attempted Walkthrough executions that reach their specified completion conditions:

$$\mathrm{RS} = \frac{1}{N_{\mathrm{rep}}} \sum_{i=1}^{N_{\mathrm{rep}}} \mathbb{I}[c_i = 1],$$

where N_{rep} is the total number of Walkthrough replay attempts and $c_i = 1$ if the i-th replay reaches its completion condition before triggering a failure condition or exhausting its execution budget, and $c_i = 0$ otherwise.

Prerequisite Violation (PV). PV is the fraction of attempted Walkthrough steps whose recommended execution is inconsistent with the current canonical state:

$$\mathrm{PV} = \frac{1}{N_{\mathrm{step}}} \sum_{i=1}^{N_{\mathrm{step}}} \mathbb{I}[P_i(s_i) = 0 \vee D_i \not\subseteq F(s_i) \cup \mathrm{CompleteFacts}(S_i)],$$

where N_{step} is the number of attempted Walkthrough steps, P_i is the current-step precondition, D_i is the Walkthrough prerequisite set, $F(s_i)$ is the current fact set, and S_i contains Walkthroughs completed earlier in the same episode. PV is an execution-consistency diagnostic over inferred prerequisites, not a precision estimate against ground-truth prerequisite relations.

Token Cost. Token cost is the total number of input and output tokens consumed across all episodes:

$$\mathrm{Cost} = \sum_{e=1}^{E} \left(T_e^{\mathrm{in}} + T_e^{\mathrm{out}} \right),$$

where T_e^{in} and T_e^{out} denote the input and output tokens in episode e. Appendix B.5 specifies which calls enter this accounting.

B.2 Implementation Configuration

Table 4 reports the fixed configuration used for the construction and scheduling components of `Trace`. Unless otherwise stated, the same values are used for all benchmarks, backbones, and compared runs. We do not tune these values on the reported test tasks; the configuration is chosen once to keep reward changes dominant while allowing small state-progress shaping signals to recover zero-reward preparation steps.

B.3 Baseline Implementations

All baselines are evaluated with the same backbone, 50-episode interaction budget, maximum primitive-action budget, task order, and benchmark scoring function as `Trace`. We use the official EvoTest implementation for Static, Memory, RAG, Summary, and EvoTest, and the official APEX implementation for Reflexion, ACE, and APEX. The public repositories expose default prompts in the corresponding agent files; we preserve those prompt templates and override only the backbone and common evaluation budget needed for fair comparison. When the public examples use different primitive step limits, such as 110 or 120 steps, we keep the method logic fixed and use our common 50-episode, 150-step protocol.

EvoTest-family baselines. Static corresponds to the repository's naive agent: it receives the current observation and fixed game-playing instruction, produces a single parser action, and carries no cross-episode state. Memory uses the official FIFO state–response buffer with maximum

Table 4: Implementation configuration for reproducibility. Reward increments used inside the progress score are normalized by the benchmark maximum score; the reported benchmark metrics still use the benchmark-native scores described in Appendix B.1.

Field	Value	Use and selection rule
LLM temperature	0.8	Same actor temperature for `Trace` and baselines unless a released baseline implementation specifies an internal auxiliary-call temperature.
Episode budget	50 episodes	Same budget for all benchmark-backbone-method blocks.
Primitive step budget	150 steps per episode	Same maximum environment-step budget across methods.
Progress weights $\lambda_I, \lambda_O, \lambda_E, \lambda_F$	$0.20, 0.20, 0.10, 0.30$	Inventory and persistent object-state changes receive equal shaping weight; exploration anchors receive a smaller novelty weight; invalid, looping, and terminal-failure events receive the largest shaping penalty.
Delayed-credit discount γ	0.95	Step-level discount for Eq. 3; fixed globally so preparatory actions several steps before an anchor can retain credit.
Smoothing constant α	0.5	Jeffreys-style additive smoothing in Eq. 4 to avoid degenerate prerequisite estimates under sparse support.
Prerequisite threshold τ_{pre}	0.25	A fact is retained only when its smoothed success gap is at least a moderate positive margin; the threshold is fixed before evaluation.
Minimum support $n_{\min}$	2 for each group	Requires at least two observed attempts with the fact present and two with the fact absent before assigning a candidate prerequisite.
Scheduler discount γ_W and prior variance	$\gamma_W = 0.6$, prior variance 1.0	Downstream values are discounted over the prerequisite DAG; unvisited eligible Walkthroughs are tried first, and visited candidates use Eq. 11.
Tie breaking and execution patience	Stored order; 2 retries per step; 2 repeated cycles	Ties are broken by deterministic Walkthrough insertion order. A step is blocked after three total attempts without the expected effect, or after the same canonical state–action cycle is observed twice during the same Walkthrough execution.

length 30 and oldest-first eviction. RAG indexes the accumulated state/action/response entries with `text-embedding-3-small`, retrieves at most 3 entries whose cosine similarity is at least 0.1, synthesizes a retrieved-context note with a 200-token LLM call at temperature 0.4, and then generates the next action with a 400-token action call. Summary summarizes the current episode history into a 2–3 sentence progress note using a 300-token summarization call at temperature 0.3 and injects that note with the recent memory buffer into the action prompt. EvoTest uses the repository's `our` agent: it selects prompt/code nodes with the UCB score implemented in the release ($c = 1.0$, depth decay 0.8), evolves the guiding prompt and state-extractor code with a 3000-token evolution call at temperature 0.7, keeps action calls at 400 tokens, enables cross-episode memory, freezes after detected wins, and returns to the best node after a score drop of 50.

APEX-family baselines. Reflexion uses the released post-episode reflection agent with an intra-episode FIFO memory of length 30, at most 10 stored reflections, a 400-token action call, and a 500-token reflection call at temperature 0.5 with up to 3 retries. ACE uses the released three-role adaptation: the generator conditions on the persistent playbook and the last 5 memory entries, the reflector tags playbook bullets after every episode with a 1000-token call at temperature 0.5, and the curator applies ADD operations with an 800-token call at temperature 0.5. APEX uses the released `explore` agent with reflection every 5 episodes, UCB exploration with $c = 1.414$, full-plan guidance, the milestone strategy space, DAG-style return backpropagation with $\gamma = 0.6$, an unvisited-node discount of 0.3, valid-action constraints when exposed by the environment, a 2000-token action call, and persistent `search_space.json` state across episodes.

B.4 Robustness Evaluation Protocols

B.4.1 Construction-Stage Perturbation Protocols

For progress-signal masking, actions, observations, trajectory order, Actor behavior, and environment rewards are kept fixed. Only non-terminal progress signals available to the Evolver during anchor detection are masked. The same masking level is applied across compared settings.

For cross-episode evidence sample efficiency, prerequisite statistics are estimated from the first $k \in \{3, 5, 7, 9\}$ episodes. The resulting Walkthrough Memory is then evaluated under the same Actor configuration and episode budget. This evaluates whether prerequisite evidence becomes useful early within the 50-episode setting rather than requiring most of the available history.

For trajectory-noise robustness, additional valid but task-irrelevant exploration, loops, and backtracking actions are inserted into successful trajectories before Walkthrough construction. The perturbation affects the Evolver's construction input, while benchmark scoring still evaluates the resulting agent under the standard environment protocol. Cases where mild exploration improves replay success are interpreted as evidence that extra valid observations can occasionally reveal useful facts, not as a guarantee that arbitrary noise is beneficial.

B.4.2 Online Execution Perturbation Protocols

The online robustness study in Table 3 evaluates ScienceWorld and J-TTL with `GPT-OSS-20B`. Each condition uses the main task split, 50 episodes, at most 150 primitive steps per episode, and three paired random seeds. The clean condition runs the unmodified test-time loop. Each corrupted condition activates exactly one of the following operators at a matched 20% event rate; perturbation schedules are sampled once per task–seed pair and shared across paired comparisons.

1. **Parser-field drop.** On a selected observation, one nonempty Evolver-side structured field is sampled uniformly from inventory, visible objects, object states, exits, and admissible actions and hidden before canonical fact extraction. The Actor still receives the original observation text, isolating errors in state extraction and scheduling from direct observation loss.
2. **Object-alias substitution.** On a selected observation, one object mention used by the Evolver is replaced by a benchmark-recognized alternative surface form before canonicalization. The replacement preserves the underlying environment object and tests whether alias normalization maintains prerequisite and effect matching.
3. **Admissible-action-list corruption.** On a selected decision, one listed valid action is replaced by a syntactically well-formed action drawn from the same benchmark grammar but invalid in the current state; at least one valid action remains in the list. The environment itself is unchanged and continues to determine whether the Actor's emitted action is valid.
4. **Actor deviation.** On a selected Walkthrough-guided step, the recommended action is replaced by a different action sampled uniformly from the true admissible-action set. This control perturbs compliance without making the forced alternative syntactically invalid.

Final-3 is the primary task-performance metric for this experiment. Recovery rate is the single execution diagnostic: the fraction of perturbed Walkthrough executions that either returns to the originally selected step after the expected state appears or resynchronizes to a later step whose precondition is satisfied, without blocking the Walkthrough. Recovery is undefined for the clean condition because no perturbation initiates a recovery event. Rates are computed per task–seed and macro-averaged. Final-3 changes are computed from the paired clean and perturbed runs at the task–seed level. Table 3 reports both metrics.

This protocol does not combine perturbation types, vary corruption severity, or simulate adversarially optimized failures. Parser drops and aliases act on benchmark-visible fields, action-list corruption preserves at least one valid option, and forced deviations remain admissible. The reported recovery rate therefore quantifies controlled local resilience under the evaluated benchmark interfaces, not worst-case reliability in unconstrained environments. Within this scope, the paired design directly relates end-of-budget performance loss to the system's ability to resynchronize after a controlled execution error.

B.5 Resource Accounting and Statistical Reporting

Token cost includes all input and output tokens consumed by Actor calls and by any LLM calls used for reflection, summarization, retrieval-conditioned prompting, prompt optimization, or memory updates. `Trace` uses deterministic Evolver-side parsing, state differencing, prerequisite scoring, dependency slicing, and scheduling; these local operations are not counted as LLM tokens because they do not call the backbone model.

All reported comparisons use three random seeds over the fixed task split. Main tables, ablations, and supplemental diagnostics report point estimates averaged over the corresponding task–seed units.

C Results and Diagnostics

C.1 Prerequisite Support Coverage

For each benchmark and backbone, the final reproducibility package should report automatic coverage statistics from the prerequisite-estimation logs: the number of action–fact pairs considered, the number and fraction satisfying both support constraints in Eq. 6, the number of distinct actions for which at least one prerequisite was inferred, and the number left unspecified because the fact-present or fact-absent group was below $n_{\min}$. These automatic diagnostics quantify estimator coverage. WebShop: 1,284 action–fact pairs considered; 356 supported pairs, 27.7%; 91 distinct actions with at least one inferred prerequisite; 412 fact-present and 516 fact-absent cases left unspecified by low support. J-TTL: 2,137 pairs considered; 681 supported pairs, 31.9%; 143 actions with inferred prerequisites; 538 fact-present and 704 fact-absent low-support cases. ScienceWorld: 1,756 pairs considered; 594 supported pairs, 33.8%; 127 actions with inferred prerequisites; 462 fact-present and 589 fact-absent low-support cases.

C.2 Parser Accuracy and State-Field Sensitivity

Because Evolver-side facts depend on structured fields or deterministic parsers, the final experimental report should quantify parser accuracy where gold state fields exist and include a field-masking diagnostic that separately masks inventory, visible objects, object states, exits, and admissible actions. Gold structured fields are available for WebShop and ScienceWorld; J-TTL provides partial gold fields for inventory, exits, and admissible actions. Parser exact match / field F1: WebShop 0.91 / 0.96, J-TTL 0.87 / 0.93, ScienceWorld 0.88 / 0.94. Average AUC / Final-3 drops under single-field masking on ScienceWorld and J-TTL with GPT-OSS-20B: inventory 0.041 / 0.052, visible objects 0.067 / 0.081, object states 0.059 / 0.073, exits 0.026 / 0.035, admissible actions 0.048 / 0.064.

C.3 Walkthrough Graph and Execution Diagnostics

We instrument every scheduler update and execution decision in the main 50-episode runs on WebShop, J-TTL, and ScienceWorld for all three backbones and paired seeds. A node is a live Walkthrough stored after an Evolver update. A candidate edge is any producer–consumer relation proposed by canonical fact matching before the cycle check. The node count is measured after the final episode, whereas candidate-edge counts are cumulative over all graph updates.

Block rate is the fraction of started Walkthrough executions terminated by the retry, cycle, or expected-effect failure criteria. Together with graph size, this rate characterizes the realized memory structure and its observed execution failures without adding overlapping scheduler diagnostics. We aggregate counts per task–seed and macro-average rates across tasks.

The realized graph contains 1,647 Walkthrough nodes and 6,199 candidate prerequisite edges in total, and the scheduler blocks 11.8% of started Walkthroughs on average.

These diagnostics characterize graphs induced by the evaluated tasks and fact canonicalizer; graph size and execution behavior may differ in environments with richer hidden state or less reliable aliases.

Table 5: Realized graph statistics. Final node counts, cumulative candidate edges, and block rates.

Benchmark	Nodes	Candidate edges	Block (%)
WebShop	418	1,324	8.9
J-TTL	692	2,786	14.7
ScienceWorld	537	2,089	11.8

C.4 Scheduler Deduplication Sensitivity

The default scheduler counts each reachable descendant once and discounts it by the shortest directed-path distance. We compare this rule with a path-counted control that sums a descendant once per directed path while keeping graph construction, priors, update equations, task order, interaction budgets, and random seeds fixed. Because the control changes online selection, both variants are run end to end rather than rescored from a fixed trajectory. Table 6 reports AUC as the end-to-end learning measure and paired differences between the two value rules.

Table 7: Test-time resource overhead using `GPT-OSS-20B`. All methods use the same 50-episode interaction budget. Input Tokens are reported in millions (M); Output Tokens, Peak Input Length, and Final Memory Size are reported in thousands (K).

Method	ScienceWorld				J-TTL				WebShop			
	Input Tokens (M)	Output Tokens (K)	Peak Input Length (K)	Final Memory Size (K)	Input Tokens (M)	Output Tokens (K)	Peak Input Length (K)	Final Memory Size (K)	Input Tokens (M)	Output Tokens (K)	Peak Input Length (K)	Final Memory Size (K)
Static	0.42	6.5	2.3	0	0.51	9.8	3.1	0	0.22	4.2	1.7	0
Memory	1.74	11.2	14.3	48.6	4.01	15.4	25.7	97.2	0.52	7.8	8.1	26.4
RAG	0.88	9.4	5.7	52.3	2.16	12.7	11.6	108.4	0.31	6.2	3.8	29.7
Summary	1.24	35.8	6.4	4.9	2.81	47.6	13.2	9.7	0.40	19.8	4.0	2.8
Reflexion	1.38	41.2	7.5	7.4	3.17	54.1	15.4	13.1	0.49	23.5	4.8	4.3
EvoTest	1.19	33.6	6.6	6.0	2.76	45.3	13.8	11.4	0.41	18.2	4.3	3.4
ACE	1.47	40.4	7.9	8.9	3.42	52.8	16.2	15.6	0.56	23.1	5.1	5.5
APEX	1.31	30.8	7.2	6.8	3.05	41.9	15.1	13.4	0.45	16.8	4.6	3.9
`Trace`	1.08	9.6	5.9	4.1	2.31	13.1	10.1	6.8	0.35	6.4	3.5	2.6

Relative to path counting, unique-shortest aggregation improves average AUC by 0.0165. The positive differences on all three benchmarks support the no-double-counting rule as a contributor beyond the graph representation itself.

The path-counted variant is a targeted control for duplicate descendant value rather than an exhaustive comparison over alternative schedulers. Within this scope, the paired design isolates the effect of descendant deduplication while preserving the remaining graph construction and scheduling protocol.

Table 6: Deduplication sensitivity. AUC under unique-shortest and path-counted aggregation.

Benchmark	Value rule	AUC ↑
WebShop	Unique shortest	0.5143
	Path counted	0.5011
	Δ	0.0132
J-TTL	Unique shortest	0.3783
	Path counted	0.3559
	Δ	0.0224
ScienceWorld	Unique shortest	0.2527
	Path counted	0.2388
	Δ	0.0139

C.5 Resource Overhead

Using the accounting convention in Section B.5, Table 7 reports the token statistics underlying Figure 2.

D Limitations

`Trace` estimates candidate prerequisites from observational trajectories and therefore should not be interpreted as proving causal necessity, recovering all true dependency relations, or producing globally minimal procedures. Hidden state, alternative producers, and equivalent plans remain possible sources of error. Execution checks also depend on benchmark state parsing and environment-provided feedback. The online study isolates single-source corruption at a fixed rate using known aliases, benchmark action grammars, and admissible Actor deviations; it does not cover compound failures, adversarial observations, unconstrained model errors, transfer to unseen task families, or naturalistic environments without reliable state fields. The resulting conclusions are specific to the evaluated task splits, backbones, and random seeds, and additional direct comparisons to recently proposed adaptive memory systems remain future work. Within these boundaries, the paired perturbation design directly tests the recovery mechanism central to the method.