APEX: AUTONOMOUS POLICY EXPLORATION FOR SELF-EVOLVING LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents have shown strong performance across a wide range of complex tasks, including interactive environments that require long-horizon decision making. But these agents cannot learn on the fly at test time. Self-evolving agents address this by accumulating memory and reflection across episodes rather than requiring model-weight updates. However, these agents often suffer from **exploration collapse**: as memory grows, behavior concentrates around familiar high-reward routines, reducing the chance of discovering better alternatives. To address this problem, we propose **A**utonomous **P**olicy **EX**ploration (**APEX**), which builds and maintains an explicit strategy space through a strategy map—a directed acyclic graph of milestones with prerequisite dependency edges. In APEX, Fork Discovery expands the map with evidence-grounded unexplored directions, while Policy Selection balances exploration and exploitation during planning. Evaluated on nine Jericho text-adventure games and WebArena, a realistic web interaction benchmark, APEX outperforms all baselines. Extensive ablations validate each component's contribution and demonstrate robustness across diverse settings, demonstrating APEX's effectiveness for sustained exploration in self-evolving agents. Code is available at `https://anonymous.4open.science/r/APEX-070F`.

1 INTRODUCTION

LLM agents can solve many complex tasks, such as software engineering Jimenez et al. (2024), scientific research automation Lu et al. (2024), and autonomous computer use Xie et al. (2024). Yet agents that start from scratch each time have a fundamental bottleneck: they do not keep useful experience across runs. As a result, they often repeat similar mistakes and fail to improve. To overcome this limitation, researchers have developed self-evolving agents Wang et al. (2025); He et al. (2025); Li et al. (2026), which use memory and reflection to carry experience across repeated attempts on the same or related tasks, enabling improvement without updating model weights.

Despite these gains, we observe exploration collapse, i.e., agents improve early, then settle into narrow behavioral patterns and stop discovering qualitatively different strategies. Figure 1 illustrates this behavior in a toy maze: a Reflexion agent achieves stable average returns but repeatedly traverses a narrow corridor and misses high-reward regions that require broader exploration. Notably, its maximum score even falls short of a static, non-learning model. Such collapse can be difficult to detect: average returns may remain high even as exploratory diversity decreases, and the agent repeatedly follows a narrow routine, foreclosing the discovery of better alternatives.

The problem is not that agents make bad choices; they never know what choices they are missing. LLM agents face an implicit, unbounded strategy space: unlike RL, there is no predefined action set, and the agent has no natural view of what it has not tried. Discovering unexplored directions requires actively constructing and maintaining an explicit map of what has been tried and what plausible alternatives remain. Two questions follow: (1) **how should the strategy space be constructed?** and (2) **which strategy should be explored next?** Existing memory and reflection mechanisms provide useful local guidance but do not maintain a persistent, explicit strategy space for decision, making it difficult to sustain broad exploration while still exploiting what has already been learned.

To address this issue, we propose **APEX** (Autonomous Policy Exploration), a framework centered on an explicit *strategy map*, a directed acyclic graph (DAG) where nodes are milestones and edges encode prerequisite dependencies. The strategy map serves as shared infrastructure for two complementary exploration mechanisms. **Fork Discovery** drives exploration at the map level: it mines past

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

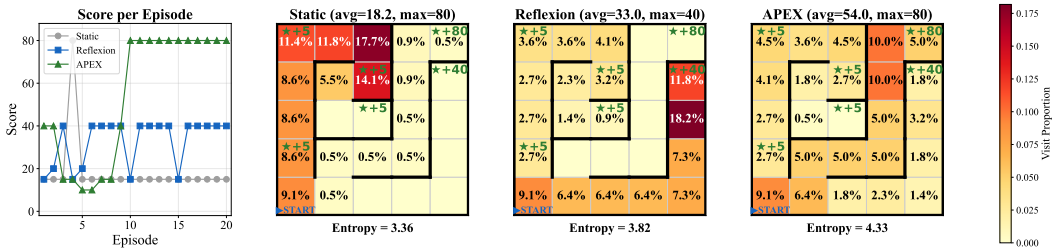


Figure 1: Illustration of exploration collapse in a maze experiment (5×5 grid, 20 episodes, 10 steps each). Room visitation heatmaps (color intensity shows visit proportion; reward cells (⋆) indicate bonus locations). Static explores broadly but inconsistently. Reflexion locks into a narrow corridor and achieves a higher average while missing high-value rooms. APEX maintains broad coverage and consistently reaches high-reward cells. APEX avoids collapse by explicitly tracking which strategies have been tried and which remain unexplored, and actively directing the agent toward unexplored directions rather than refining familiar ones.

episodes for directions the agent observed but never pursued, and expands the map's frontier with new milestones. **Policy Selection** drives exploration at the episode level: it scores milestones via principled uncertainty estimates, ensuring the agent visits under-explored directions rather than repeatedly exploiting familiar ones. Together, they turn exploration into a closed loop—Fork Discovery continuously opens new territory, and Policy Selection ensures the agent visits it.

We evaluate APEX on nine text adventure games from the Jericho benchmark Hausknecht et al. (2020) and on WebArena Zhou et al. (2024), a realistic web interaction benchmark. APEX consistently outperforms all baselines across both settings, with the largest gains on tasks that require discovering qualitatively different strategies—exactly the regime where exploration collapse is most harmful. Ablations and case studies trace these gains to the strategy map: Fork Discovery uncovers directions that reflection-based methods never consider, and Policy Selection ensures they are systematically visited. The results hold robustly across hyperparameter choices and backbone LLMs, underscoring the generality of APEX.

2 Related Work

Self-Evolving LLM Agents. Self-evolving agents improve through accumulated experience across episodes rather than updating model weights. Early work introduced verbal self-reflection Shinn et al. (2023), experience distillation Wang et al. (2025), and growing skill libraries Wang et al. (2024). More recent work scales these ideas further. ReasoningBank Ouyang et al. (2025) distills reasoning strategies from both successful and failed experiences into structured memory. EvoTest He et al. (2025) introduces a gradient-free evolutionary framework that revises an agent's prompt, memory, and tools between episodes. JitRL Li et al. (2026) adjusts the action logits of the current episode by retrieving relevant historical trajectories from memory, enabling continual improvement without gradient updates. However, none of these systems builds an explicit strategy space to guide exploration; this is the gap APEX addresses.

Exploration in Fixed Action Spaces. Classical exploration methods assume a predefined primitive action space. They fall into two broad families: *intrinsic motivation*, which rewards agents for visiting novel states or exhibiting diverse behaviors, including count-based bonuses Tang et al. (2017); Choi et al. (2019), curiosity-driven rewards Li et al. (2019); Bougie & Ichise (2021), episodic curiosity Badia et al. (2020), and diversity-maximizing skill discovery Eysenbach et al. (2019); and *planning under uncertainty*, which models the agent's uncertainty to guide exploration, including UCB Auer et al. (2002); Zheng et al. (2025), Thompson Sampling Thompson (1933); Russo et al. (2018), and maximum entropy objectives Haarnoja et al. (2018). Despite their variety, all share a fundamental assumption: the primitive action space is fixed and given. They answer the question of which known option to try next, but never ask what options should exist in the first place. APEX addresses this complementary question: Fork Discovery actively constructs the strategy space from trajectory evidence, while Policy Selection operates over it.

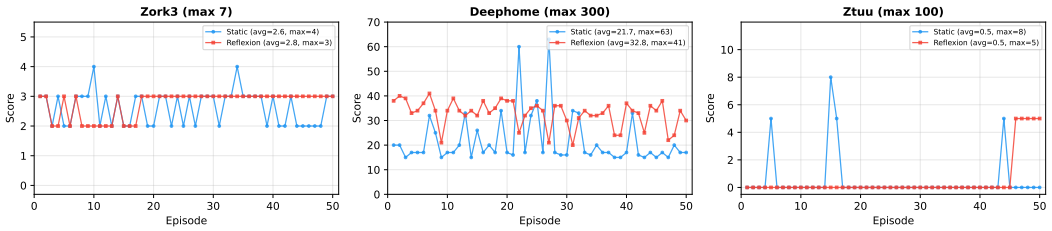


Figure 2: Comparison of Static and Reflexion on three Jericho games. Reflexion often achieves a higher average but a lower maximum than Static, indicating convergence to a narrow behavioral distribution that misses superior strategies.

Entropy Bottleneck in LLM Reinforcement Learning. Recent work identifies entropy collapse in LLM Reinforcement Learning (RL). Cui et al. (2025) show that RL training systematically reduces policy entropy, causing the agent to concentrate probability mass on familiar high-reward actions and progressively lose behavioral diversity. Yue et al. (2025) show that RLVR improves average performance but narrows the reasoning boundary, as RL sharpens the distribution over existing paths rather than discovering new ones. These works focus on the weight-update regime, while APEX addresses the analogous collapse in test-time self-evolving agents.

3 Exploration Collapse: An Empirical Study

We study how Reflexion Shinn et al. (2023), a representative self-evolving agent, and a memoryless Static baseline perform across multiple tasks: a 5×5 maze with 20 episodes of 10 steps each, and three text adventure games, Zork3, Deephome, and Ztuu, with 50 episodes of 120 steps each. Both use Gemini 3 Flash as the backbone LLM with identical settings. In the maze, the agent receives a localized text observation describing the current room and its available exits, without global coordinates. The action space consists of the cardinal directions reachable from the current cell.

As shown in Figure 1,Reflexion converges to a fixed route that reliably reaches the +40 reward but never discovers the +80 cell, while Static occasionally reaches the +80 cell. The heatmap of Reflexion shows heavy concentration in that corridor. APEX, by contrast, explores the maze broadly and consistently reaches the high-reward cell. As shown in Figure 2, the same pattern holds across three text adventure games of varying difficulty: in most cases, Reflexion achieves a higher average but a lower maximum than Static.

Once Reflexion finds a rewarding strategy, the agent repeats it episode after episode and stops exploring different approaches. Trapped in a local optimum, it never discovers the superior strategies that Static occasionally stumbles upon through sheer behavioral diversity. This is **exploration collapse**: the agent converges to a narrow behavioral routine and stops searching for better alternatives. We provide additional controlled-maze experiments across seven backbones in Appendix F.

4 Method: APEX

Overview. As shown in Figure 3, APEX addresses exploration collapse by maintaining an explicit *strategy map* that records what has been tried and what plausible alternatives remain. For example, if prior trajectories have identified three milestones (e.g., three doors), the map should direct the agent toward discovering novel alternatives (e.g., a fourth door), as well as revisiting under-explored ones.

Strategy Map. The strategy map $\mathcal{G} = (V, E, \theta)$ is a Directed Acyclic Graph (DAG) where each node $v \in V$ is a milestone (a discrete sub-goal described in natural language with associated key actions, e.g., "collect the brass lantern" via go north, take lantern), and each edge $(u, v) \in E$ encodes that u must be achieved before v. Each node carries statistics $\theta_v = (n(v), \bar{r}(v), \mathrm{Var}(v))$: visit count, average reward, and reward variance. A designated root node $v_0 \in V$ represents the initial state. Milestones with no prerequisites have v_0 as their sole dependency. At initialization, the map contains only v_0 and is subsequently populated with milestones and edges extracted from experience.

The map defines a higher-level planning problem. The abstract state $s \subseteq V$ is the set of milestones achieved so far in the current episode. We denote by $\mathcal{A}(s)$ the structurally eligible milestones—those not yet achieved whose predecessors are all in s:

$$\mathcal{A}(s) = \big\{ v \in V \setminus s : \{u \mid (u, v) \in E\} \subseteq s \big\}. \tag{1}$$

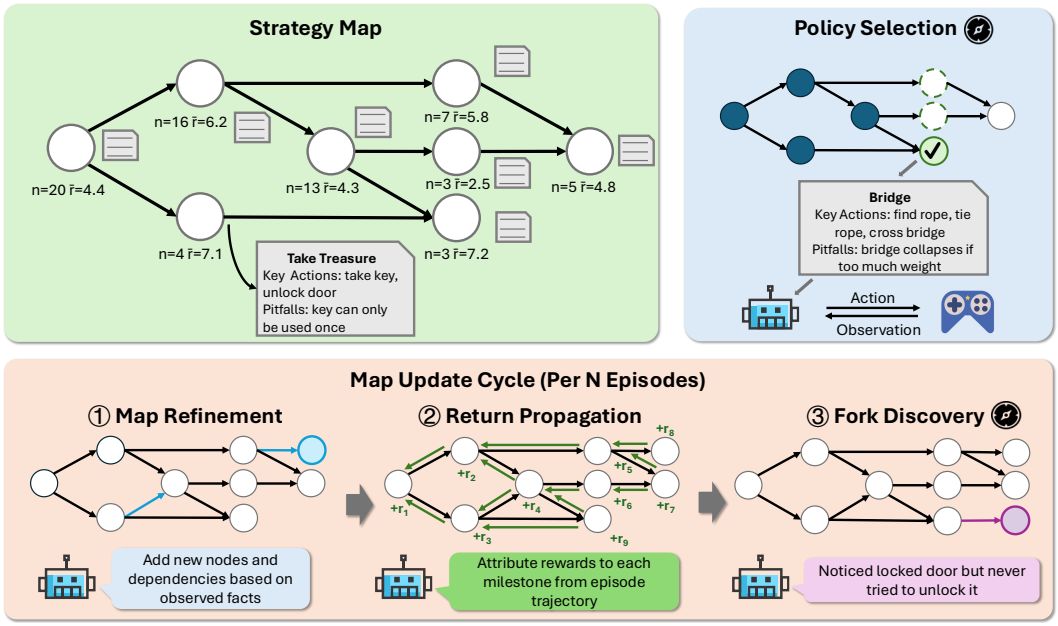


Figure 3: Overview of APEX. Each node in the strategy map displays n (visit count) and $\bar{r}$ (average reward). Within each episode, **Policy Selection** is invoked repeatedly to choose the next milestone. When a milestone is achieved or failed, Policy Selection is called again until the episode ends. Every N episodes, a periodic reflection cycle runs **Map Refinement** to update existing nodes and edges, **Return Propagation** to update node statistics on the refined map, and **Fork Discovery** to expand the map with new milestones and dependencies.

We formalize this as a Milestone Abstract MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$, where $\mathcal{S}$ is the set of all valid abstract states, $\mathcal{A}(s)$ is the eligible milestone set defined above, $\mathcal{T}(s \cup \{v\} \mid s, v)$ is the transition probability from s to $s \cup \{v\}$ upon attempting milestone v, and $R(s, v)$ is the distribution of rewards upon attempting v, which we approximate using $\bar{r}(v)$.

4.1 Policy Selection

Policy Selection uses $\mathcal{A}(s, b) = \mathcal{A}(s) \setminus b$, where b contains milestones failed or skipped in the current episode. This reduces selection to a finite bandit problem over DAG-structured candidates. Each eligible milestone $v \in \mathcal{A}(s, b)$ is scored by a function f of its bandit statistics, and the highest-scoring one is selected:

$$\text{score}(v) = f\big(n(v),\, \bar{r}(v),\, \text{Var}(v)\big), \qquad v^* = \arg\max_{v \in \mathcal{A}(s,b)} \text{score}(v). \tag{2}$$

The choice of f determines the exploration-exploitation tradeoff. Three bandit instantiations are:

- **Thompson Sampling** Thompson (1933); Russo et al. (2018): $\text{score}(v) = \text{sample}\Big(\mathcal{N}\Big(\bar{r}(v),\, \sqrt{\text{Var}(v)/n(v)}\Big)\Big)$. High-variance milestones remain under exploration even when their average reward is low.
- **UCB** Auer et al. (2002): $\text{score}(v) = \bar{r}(v) + c\sqrt{\log\big(\sum_{u \in \mathcal{A}(s,b)} n(u)\big)/n(v)}$, where $\sum_{u \in \mathcal{A}(s,b)} n(u)$ is the total visit count over all currently eligible milestones. The bonus decreases as a milestone is visited more, naturally shifting focus toward less-explored candidates.
- **ϵ-Greedy** Sutton & Barto (2018): selects the highest $\bar{r}(v)$ milestone with probability $1 - \epsilon$, and a random eligible milestone with probability ϵ.

We use the variance-aware Thompson-style sampler as the default; its prior scale and variance floor are specified in Appendix C. Eligible milestones with $n(v) = 0$ receive $+\infty$ scores and priority over visited ones. Statistics are updated by the periodic Return Propagation step (Section 4.3).

Policy Selection proceeds sequentially during episode execution: after each milestone is achieved or fails, we score all milestones whose dependencies have been actually achieved, select the highest-scoring eligible one, and execute based on the milestone. This is more robust than fixing a full milestone sequence upfront: if the plan were fixed and a milestone fails, the agent would still attempt its dependents even though their prerequisites were never met. With sequential selection, failed milestones never satisfy their dependents' prerequisites, so those dependents never enter $\mathcal{A}(s)$, and the agent naturally continues with other reachable branches. If a milestone is skipped, the agent does not retry it during the same episode. Skipping does not count as completion, so milestones requiring it remain unavailable.

4.2 Map Refinement

After each episode t, an LLM generates a structured summary $\tilde{h}_t$ from the raw trajectory h_t:

$$\tilde{h}_t = \mathrm{LLM}_{\text{sum}}(h_t). \tag{3}$$

Every N episodes, the reflection cycle operates on $\tilde{H}_t = \{\tilde{h}_{t-N+1}, \ldots, \tilde{h}_t\}$, the summaries from the most recent N episodes. As the first step of the reflection cycle, Map Refinement updates existing nodes based on observed facts from recent episodes. It refines key actions when better sequences are discovered, corrects dependency edges when assumed prerequisites turn out to be wrong, and prunes duplicate nodes. Formally, the LLM receives $\mathcal{G}_t$ and $\tilde{H}_t$ and returns:

$$(\Delta V^M,\, \Delta E^M) = \mathrm{LLM}_{\text{map}}(\mathcal{G}_t,\, \tilde{H}_t), \tag{4}$$

where ΔV^M and ΔE^M are node and edge edit operations, including fact-grounded additions, attribute updates, duplicate-node deletions, and dependency replacements. Applying these edits gives $\mathcal{G}'_t = \mathrm{Apply}(\mathcal{G}_t; \Delta V^M, \Delta E^M) = (V'_t, E'_t, \theta'_t)$. The operator preserves the statistics of retained nodes and discards those of deleted duplicates (Appendix C). The key distinction from Fork Discovery is intent: Map Refinement encodes observed facts and corrects errors, while Fork Discovery adds nodes grounded in unexplored possibilities.

4.3 Return Propagation

Every N episodes, after Map Refinement, Return Propagation updates node statistics in $\mathcal{G}'_t$ using $\tilde{H}_t = \{\tilde{h}_{t-N+1}, \ldots, \tilde{h}_t\}$, the summaries of the most recent N episodes. For each episode summary $\tilde{h}_i \in \tilde{H}_t$, an LLM attributes a scalar reward $r(v)$ to each milestone v attempted in that episode based on the observed score gains:

$$\{r(v)\}_{v \in V_a} = \mathrm{LLM}_{\text{reward}}(\tilde{h}_i, \mathcal{G}'_t). \tag{5}$$

Let V_a denote the set of milestones attempted in episode i. For each $v \in V_a$, we compute a credit score recursively along the DAG edges:

$$G_v = r(v) + \gamma \sum_{u \in \mathrm{succ}(v) \cap V_a} G_u, \tag{6}$$

where $\mathrm{succ}(v)$ denotes the direct successors of v in $\mathcal{G}$, and $\gamma \in [0, 1]$ controls how much downstream credit propagates back. This recursion is computed via a reverse topological pass, ensuring that v only receives credit from milestones it directly enables, not from parallel branches. For each $v \in V_a$, $n(v)$ is incremented and $\bar{r}(v)$ is updated as a running mean over G_v values. Here, $n(v)$ includes unsuccessful attempts, and $\mathrm{Var}(v)$ is estimated from the same attributed-return samples as $\bar{r}(v)$. Milestones that were never scheduled in the episode are skipped and their statistics remain unchanged. This gives credit to enabling milestones that yield little reward on their own but unlock high-reward successors (e.g., "obtain the key" enables "unlock the treasure room").

4.4 Fork Discovery

Return Propagation refines what the agent knows about existing milestones, but it cannot discover what it has never tried. As the agent accumulates experience, the map must grow to cover new territory. To actively push the map's frontier outward, we introduce **Fork Discovery**, which generates new milestones by combining the current map with information observed during gameplay, such as available actions and unexplored directions that the agent has yet to pursue.

Every N episodes, an LLM receives $\mathcal{G}'_t$ and $\tilde{H}_t$, identifies actions the agent observed or had available but never pursued, and proposes new milestones with dependency edges:

$$(\Delta V^F,\ \Delta E^F) = \mathrm{LLM}_{\text{fork}}(\mathcal{G}'_t,\ \tilde{H}_t). \tag{7}$$

After all three steps, the updated map is:

$$\mathcal{G}_{t+1} = (V'_t \cup \Delta V^F,\ E'_t \cup \Delta E^F,\ \theta_{t+1}), \tag{8}$$

where each new node $v \in \Delta V^F$ is initialized with $n(v) = 0$.

Together, the four components form a coherent system: Fork Discovery and Policy Selection drive exploration: the former expands the map's frontier and the latter directs the agent within it. Map Refinement and Return Propagation keep the map accurate and its statistics current. As the strategy map matures, Fork Discovery is disabled after a configurable episode threshold, naturally shifting the system from exploration toward exploitation. The complete pseudocode is given in Appendix A. See Appendix C for implementation details and prompt templates.

5 Experiments

We design our experiments to answer the following four research questions:

- **RQ1**: How does APEX perform compared to state-of-the-art baselines?
- **RQ2**: How do the strategy map structure, Fork Discovery, and Policy Selection each contribute to APEX's performance?
- **RQ3**: What does APEX actually discover, and why does it outperform baselines?
- **RQ4**: How sensitive is APEX to key hyperparameters such as the reflection interval and the choice of backbone LLM?

5.1 Setup

Benchmarks. We evaluate on two environments to test APEX across different task types:

- **Jericho** Hausknecht et al. (2020): A benchmark suite for interactive fiction games where agents interact purely via textual commands. We evaluate on nine games spanning diverse challenges: Zork1, Balances, Zork3, Temple, Pentari, Ludicorp, Deephome, Detective, and Ztuu.
- **WebArena** Zhou et al. (2024): A realistic web environment comprising multiple functional websites. It challenges agents to navigate complex interfaces and execute sequential actions to fulfill user instructions. We evaluate on all 812 tasks across five website categories; each task is attempted for 5 episodes.

Metric. We report **Final-K**: the mean performance over the final K episodes. For Jericho, K=5 over 50 episodes per game. For WebArena, K=3 over 5 episodes per task.

Baselines. We evaluate five baselines: (1) **Static**: a non-learning agent with no cross-episode memory. (2) **Memory**: an in-context learning baseline that accumulates past episode transcripts into the agent's context window, using a FIFO strategy to handle token limits. (3) **Reflexion** Shinn et al. (2023): a prompt-based method where the agent generates structured textual self-reflections after each episode to guide subsequent attempts. (4) **EvoTest** He et al. (2025): an evolutionary framework that iteratively revises the agent's prompt, memory, and tools between episodes, but without maintaining an explicit strategy map. (5) **ACE** Zhang et al. (2025): a framework that treats contexts as evolving playbooks, accumulating and refining strategies through a modular grow-and-refine process, without maintaining an explicit strategy map. All methods, including APEX, use Gemini 3 Flash as the backbone LLM with identical settings. Detailed baseline descriptions are in Appendix B.

5.2 Main Results

We address **RQ1** by comparing APEX against all baselines across Jericho and WebArena.

Jericho Results. As shown in Table 1, APEX achieves the best Final-5 on all nine games. Figure 4 shows per-episode learning curves on three representative games; full curves for all nine games are in Appendix D. The gains are most pronounced in games that require discovering qualitatively different strategies across episodes, where the strategy map enables systematic exploration beyond what

Table 1: Results on Jericho games. Final-5: mean score of episodes 46–50.

Method	Zork1	Balances	Zork3	Temple	Pentari	Ludicorp	Deephome	Detective	Ztuu
Static	39.8	11.0	2.8	6.2	6.0	13.6	17.2	70.0	0.0
Memory	46.0	13.0	2.4	6.2	7.0	17.0	29.0	130.0	0.0
Reflexion	59.8	20.0	3.0	9.0	10.0	35.0	29.6	144.0	5.0
EvoTest	42.0	18.0	2.4	10.0	16.0	48.0	56.8	314.0	25.0
ACE	46.0	18.0	2.2	7.2	19.0	19.2	8.4	198.0	3.4
APEX (Ours)	**73.0**	**23.0**	**4.6**	**20.8**	**46.0**	**58.8**	**103.4**	**322.0**	**29.0**

Table 2: Results on WebArena (Final-3)

Method	Shopping	Reddit	GitLab	CMS	Map
Static	34.7	45.7	36.3	44.8	56.3
Memory	39.1	53.8	53.9	53.2	55.2
Reflexion	34.1	36.2	50.8	46.7	54.2
EvoTest	38.9	50.9	54.4	53.8	52.9
ACE	35.9	52.5	51.6	52.4	55.7
APEX	**42.9**	**68.2**	**60.2**	**55.1**	**57.6**

Table 3: Cross-task generalization on WebArena.

Method	Shopping	Reddit	GitLab	CMS	Map
Static	33.9	44.5	35.9	43.9	49.2
Memory	37.5	52.7	53.2	**53.1**	55.1
Reflexion	32.8	34.9	49.5	46.2	48.6
EvoTest	37.5	47.7	53.4	52.7	50.4
ACE	26.6	46.5	50.5	48.9	51.4
APEX	**41.7**	**56.6**	**55.9**	52.7	**56.0**

reflection or memory alone can achieve. APEX outperforms all baselines, showing that structured exploration is beneficial even in games with strong single-path strategies. Overall, the results suggest that structured strategy spaces are most valuable when the task rewards diversity of exploration rather than depth along a single route.

WebArena Results. We further evaluate on WebArena, where agents interact with structured web pages. As shown in Table 2, APEX achieves the best Final-3 score and leads on all five categories, outperforming the next best baseline by a clear margin. The gains are most pronounced on Reddit and GitLab, where tasks require multi-step sequences with non-obvious intermediate actions—precisely the setting where Fork Discovery adds value by encoding untried action paths into the strategy map. Overall, APEX's structured exploration mechanism transfers effectively beyond text adventure games to realistic web environments. We report independent-run results on selected Jericho and WebArena tasks in Appendix G.

Cross-Task Generalization. Table 3 evaluates cross-task generalization on WebArena, where each task is attempted without any task-specific prior experience. Performance therefore reflects only what the agent has learned from other tasks. APEX leads across the majority of domains and achieves the best overall score, showing that the structured knowledge encoded in the strategy map transfers effectively to unseen tasks rather than merely memorizing task-specific solutions.

5.3 Ablation Studies

We address **RQ2** by isolating each design choice (the map structure, space expansion method, and selection algorithm), varying one at a time while keeping the rest fixed at the default configuration: DAG strategy map, Thompson Sampling, Fork Discovery, and reflect interval $N{=}5$.

Strategy Space Representation: DAG vs. Flat List. We compare two representations: (1) **Flat List**, milestones maintained as an unordered list without dependency edges, and (2) **DAG (Ours)**, milestones connected by multi-prerequisite dependency edges. As shown in Table 4, DAG achieves the best Final-5 on all three games. Flat List underperforms DAG across all games: without dependency structure, the agent cannot reason about milestone ordering or reachability, limiting effective exploration. The results confirm that capturing multi-prerequisite dependencies is key to the strategy map's effectiveness. We provide an additional comparison with a retrievable skill bank in Appendix I.

Fork Discovery: FD vs. No FD. The strategy map must grow as the agent accumulates experience, but whether to explore matters. We compare two approaches: No FD updates the map only with milestones the agent has directly visited, reinforcing known paths without identifying unexplored directions; Fork Discovery, by contrast, mines past episodes for alternatives the agent observed but never pursued, actively expanding the map's frontier. As shown in Table 5, without Fork Discovery, the agent can only reinforce what it has already tried; the gap is smaller on Zork1 but large on Ztuu,

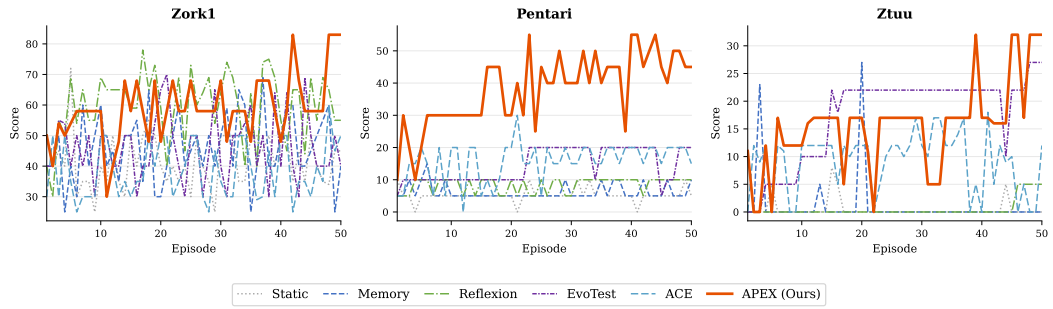


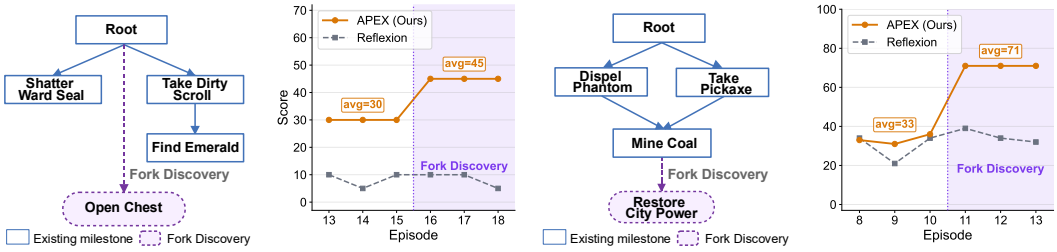
Figure 4: Per-episode scores on three representative Jericho games.

Table 4: Strategy space representation comparison (Final-5).

Representation	Zork1	Pentari	Ztuu
Flat List	52.1	27.0	12.0
DAG (Ours)	**73.0**	**46.0**	**29.0**

Table 5: Fork Discovery ablation (Final-5).

Method	Zork1	Pentari	Ztuu
No FD	65.0	24.0	5.0
FD (Ours)	**73.0**	**46.0**	**29.0**



(a) Pentari: Map (b) Pentari: Score (c) Deephome: Map (d) Deephome: Score

Figure 5: Fork Discovery case studies on Pentari (a,b) and Deephome (c,d): strategy maps and per-episode scores before and after Fork Discovery adds a new milestone.

where higher scores require discovering directions the agent has never pursued. Fork Discovery closes this gap by actively expanding the map's frontier as the agent gains experience.

Policy Selection: Thompson vs. UCB vs. ϵ-Greedy. We compare three selection strategies: ϵ-Greedy ignores uncertainty and selects randomly with probability ϵ; UCB adds a count-based optimism bonus toward less-visited milestones; Thompson Sampling draws from each milestone's posterior, naturally favoring high-uncertainty ones. As shown in Table 6, both UCB and Thompson outperform ϵ-Greedy, confirming that leveraging uncertainty is more effective than random exploration. Thompson outperforms UCB across all three games, with the largest gap on Ztuu where variance-awareness provides the most benefit.

Return Propagation: DAG vs. Sequential. We compare two credit assignment schemes for Return Propagation: **Sequential** discounts rewards along the episode path order: given the ordered sequence $(v_1, \ldots, v_k)$ of attempted milestones, $G_{v_i} = r(v_i) + \gamma \cdot G_{v_{i+1}}$, so each milestone inherits credit from all later milestones regardless of dependency structure. **DAG** instead propagates credit strictly along prerequisite edges: $G_v = r(v) + \gamma \sum_{u \in \mathrm{succ}(v) \cap V_a} G_u$, ensuring a milestone only receives credit from milestones it directly enables. As shown in Table 8, DAG credit assignment outperforms Sequential across all three games. Sequential conflates credit across parallel branches, assigning reward to milestones that did not causally enable it. DAG restricts credit flow to actual prerequisite edges, producing more accurate value estimates and better-informed Policy Selection.

5.4 Analysis: What Does APEX Discover?

We address **RQ3** by inspecting the strategy maps APEX builds in practice. Figure 5 shows two representative Fork Discovery events and their score impact.

Table 6: Policy Selection ablation (Final-5).

Method	Zork1	Pentari	Ztuu
ϵ-Greedy	48.0	35.0	1.0
UCB	59.7	43.0	12.0
Thompson (Ours)	**73.0**	**46.0**	**29.0**

Table 7: Reflect interval N sensitivity (Final-5).

N	Zork1	Pentari	Ztuu
1	34.6	16.0	25.0
2	**73.0**	26.0	24.0
5	70.2	**46.0**	**29.0**
10	52.8	30.0	22.0

Table 8: Return Propagation ablation.

Method	Zork1	Pentari	Ztuu
Sequential	71.6	35.0	17.0
DAG (Ours)	**73.0**	**46.0**	**29.0**

Table 9: Comparison of cost ($) among all the methods.

	Static	Memory	Reflexion	EvoTest	ACE	APEX
Cost	278	320	306	306	324	318

In **Pentari**, Fork Discovery identifies "Open Chest", a milestone the agent had encountered but never deliberately attempted, and adds it directly to the strategy map. Scores jump from an average of 30 to 45, a level Reflexion never reaches. In **Deephome**, Fork Discovery encodes "Restore City Power", which requires both "Dispel Phantom" and "Take Pickaxe" via "Mine Coal". Scores jump from 33 to 71, while Reflexion plateaus at 30–40 throughout. Both cases show that Fork Discovery surfaces valuable directions the agent had evidence for but never explicitly pursued.

5.5 Sensitivity Analysis

We address **RQ4** by examining how APEX's performance varies with the reflection interval and the choice of backbone LLM.

Reflect Interval. Fork Discovery and Map Refinement run periodically every N episodes. A smaller N means more frequent map updates but higher LLM cost, while a larger N means less overhead but slower adaptation. As shown in Table 7, we evaluate the sensitivity to this parameter across three games. Across all three games, $N = 5$ and $N = 2$ perform well, suggesting that moderate reflection intervals strike the right balance between map accuracy and stability. $N = 1$ consistently underperforms: reflecting after every single episode provides too little new experience per cycle, leading to unstable map updates. Very large N also degrades performance, as infrequent updates cause the map to lag behind the agent's accumulated experience.

LLM Backbone. APEX consistently outperforms all baselines across three backbone LLMs, showing that its gains are not tied to any particular model. Full results are in Appendix D.2.

5.6 Cost Analysis

Table 9 reports the API cost for each method on WebArena (812 tasks, 5 episodes per task). APEX's cost is comparable to other methods despite running additional reflection components, because milestone-guided prompts remain concise throughout training and naturally bound context growth. We also analyze the storage overhead of the strategy map in Appendix J.

6 Conclusion

We showed that self-evolving LLM agents often suffer from exploration collapse, converging to narrow behavioral patterns and missing superior strategies. APEX addresses the root cause: without an explicit strategy space, agents have no way to know what they have not tried. The strategy map provides this structure, Fork Discovery expands it, and Policy Selection navigates it. The result is consistent improvement over all baselines across nine Jericho games and WebArena, with ablations supporting the evaluated design choices.

Limitations. The strategy map's quality is bounded by the LLM's ability to identify milestones and infer unexplored directions, subtle or deeply hidden game mechanics may be overlooked by Fork Discovery. Moreover, APEX explore directions grounded in observed trajectories, and it cannot discover strategies that lie entirely outside the agent's past knowledge. Our evaluation focuses on text adventure games and web interaction tasks where progress naturally decomposes into discrete milestones. Tasks with continuous or less structured progress signals may require adapting how milestones are defined and evaluated.

AI USE STATEMENT

In this work, we used OpenAI Codex to assist with implementing parts of the codebase. We manually develop the proposed method, design experiments, and interpret experimental results. We used Google Gemini to assist with manuscript drafting and revision, readability improvements, reference checking, and formatting. The authors carefully reviewed all AI-assisted work, including the code and manuscript revisions, and checked the mathematical derivations for correctness. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the assistance of generative AI.

ETHICS STATEMENT

This work evaluates LLM agents in the Jericho and WebArena benchmark environments and does not involve human subjects or the collection of sensitive personal data. More capable autonomous exploration may nevertheless create misuse risks if deployed on real websites, including unauthorized interaction, scraping, or resource consumption. Deployment outside controlled benchmarks should therefore use appropriate access controls, rate limits, monitoring, and human oversight.

REPRODUCIBILITY STATEMENT

The method, training-free workflow, evaluation protocol, and principal hyperparameters are described in Sections 4 and 5. Appendix A provides the complete workflow; Appendix C reports implementation details and prompts; and Appendix D.1 and the subsequent supplementary sections provide additional results and statistical definitions. The code repository linked in the abstract contains the implementation and experimental configurations.

REFERENCES

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, 2002. doi: 10.1023/A:1013689704352. URL https://doi.org/10.1023/A:1013689704352.

Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andrew Bolt, and Charles Blundell. Never give up: Learning directed exploration strategies. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=Sye57xStvB.

Nicolas Bougie and Ryutaro Ichise. Fast and slow curiosity for high-level exploration in reinforcement learning. *Appl. Intell.*, 51(2):1086–1107, 2021. doi: 10.1007/S10489-020-01849-3. URL https://doi.org/10.1007/s10489-020-01849-3.

Jongwook Choi, Yijie Guo, Marcin Moczulski, Junhyuk Oh, Neal Wu, Mohammad Norouzi, and Honglak Lee. Contingency-aware exploration in reinforcement learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HyxGB2AcY7.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng, Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and Ning Ding. The entropy mechanism of reinforcement learning for reasoning language models. *CoRR*, abs/2505.22617, 2025. doi: 10.48550/ARXIV.2505.22617. URL https://doi.org/10.48550/arXiv.2505.22617.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=SJx63jRqFm.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Proceedings of Machine Learning Research, pp. 1856–1865. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

Matthew J. Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 7903–7910. AAAI Press, 2020. doi: 10.1609/AAAI.V34I05.6297. URL https://doi.org/10.1609/aaai.v34i05.6297.

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. *CoRR*, abs/2510.13220, 2025. doi: 10.48550/ARXIV.2510.13220. URL https://doi.org/10.48550/arXiv.2510.13220.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Boyao Li, Tao Lu, Jiayi Li, Ning Lu, Yinghao Cai, and Shuo Wang. Curiosity-driven exploration for off-policy reinforcement learning methods. In *2019 IEEE International Conference on Robotics and Biomimetics, ROBIO 2019, Dali, China, December 6-8, 2019*, pp. 1109–1114. IEEE, 2019. doi: 10.1109/ROBIO49542.2019.8961529. URL https://doi.org/10.1109/ROBIO49542.2019.8961529.

Yibo Li, Zijie Lin, Ailin Deng, Xuan Zhang, Yufei He, Shuo Ji, Tri Cao, and Bryan Hooi. Just-in-time reinforcement learning: Continual learning in LLM agents without gradient updates. *CoRR*, abs/2601.18510, 2026. doi: 10.48550/ARXIV.2601.18510. URL https://doi.org/10.48550/arXiv.2601.18510.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob N. Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *CoRR*, abs/2408.06292, 2024. doi: 10.48550/ARXIV.2408.06292. URL https://doi.org/10.48550/arXiv.2408.06292.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. *CoRR*, abs/2509.25140, 2025. doi: 10.48550/ARXIV.2509.25140. URL https://doi.org/10.48550/arXiv.2509.25140.

Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *Found. Trends Mach. Learn.*, 11(1):1–96, 2018. doi: 10.1561/2200000070. URL https://doi.org/10.1561/2200000070.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction, 2nd Edition*. MIT Press, 2018. URL http://www.incompleteideas.net/book/the-book-2nd.html.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 2753–2762, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3a20f62a0af1aa152670bab3c602feed-Abstract.html.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

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. *Trans. Mach. Learn. Res.*, 2024, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference*

on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *CoRR*, abs/2504.13837, 2025. doi: 10.48550/ARXIV.2504.13837. URL https://doi.org/10.48550/arXiv.2504.13837.

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. *CoRR*, abs/2510.04618, 2025. doi: 10.48550/ARXIV.2510.04618. URL https://doi.org/10.48550/arXiv.2510.04618.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/zheng25o.html.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

A Complete Workflow

Algorithm 1 presents the complete APEX workflow. Here $\tilde{h}_t$ denotes the LLM-generated structured summary of episode t, and $\tilde{H}_t = \{\tilde{h}_i \mid t - N + 1 \le i \le t\}$ denotes the summaries of the most recent N episodes used by the reflection operations.

Algorithm 1 APEX: Core Workflow

Require: Strategy map $\mathcal{G} = (V, E, \theta)$, episodes T, reflect interval N, discount γ

for $t = 1$ to T **do**
  $s \leftarrow \{v_0\}, \quad b \leftarrow \emptyset, \quad V_a \leftarrow \emptyset, \quad h_t \leftarrow []$
  while episode not ended **do**
    $\mathcal{A}(s, b) \leftarrow \{v \in V \setminus (s \cup b) : \{u \mid (u, v) \in E\} \subseteq s\}$
    $v^* \leftarrow \arg\max_{v \in \mathcal{A}(s,b)} f\big(n(v), \bar{r}(v), \mathrm{Var}(v)\big)$ // score via Thompson Sampling
    Execute v^*; observe $(o, r(v^*))$; $h_t \leftarrow h_t \cup \{(v^*, o, r(v^*))\}$; $V_a \leftarrow V_a \cup \{v^*\}$
    if v^* succeeded **then**
      $s \leftarrow s \cup \{v^*\}$
    else
      $b \leftarrow b \cup \{v^*\}$ // failed or auto-skipped; does not satisfy prerequisites
    end if
  end while
  $\tilde{h}_t \leftarrow \mathrm{LLM}(h_t)$
  if $t \bmod N = 0$ **then**
    $\tilde{H}_t \leftarrow \{\tilde{h}_{t-N+1}, \ldots, \tilde{h}_t\}$
    $(\Delta V^M, \Delta E^M) \leftarrow \mathrm{LLM}(\mathcal{G}_t,\ \tilde{H}_t)$ // Map Refinement
    $\mathcal{G}'_t = (V'_t, E'_t, \theta'_t) \leftarrow \mathrm{Apply}(\mathcal{G}_t; \Delta V^M, \Delta E^M)$
    for each episode $i \in \{t - N + 1, \ldots, t\}$, $v \in V_a^i$ in reverse topological order **do**
      $G_v \leftarrow r(v) + \gamma \sum_{u \in \mathrm{succ}(v) \cap V_a^i} G_u$ // Return Propagation
      $n(v)$ += 1; $\quad \bar{r}(v) \leftarrow \bar{r}(v) + \frac{1}{n(v)}\big(G_v - \bar{r}(v)\big)$
      Update $\mathrm{Var}(v)$ from the same return samples G_v
    end for
    $(\Delta V^F, \Delta E^F) \leftarrow (\emptyset, \emptyset)$
    if Fork Discovery is enabled by the freeze schedule **then**
      $(\Delta V^F, \Delta E^F) \leftarrow \mathrm{LLM}(\mathcal{G}'_t,\ \tilde{H}_t)$ // Fork Discovery
    end if
    $\mathcal{G}_{t+1} \leftarrow (V'_t \cup \Delta V^F,\ E'_t \cup \Delta E^F,\ \theta_{t+1})$
  end if
end for

B Baseline Descriptions

We describe the five baselines used in our experiments.

Static. The Static baseline is a memoryless agent that begins each episode from scratch with no knowledge of previous runs. It receives only the current episode's observations and a fixed system prompt, with no reflection, no accumulated memory, and no cross-episode information of any kind. Static serves as a lower bound: its occasional successes arise purely from behavioral diversity across independent episodes, not from any learning signal. Because it never exploits past experience, it cannot improve over time—but for the same reason, it never collapses into a narrow behavioral routine.

Memory. The Memory baseline is an in-context learning agent that accumulates raw episode transcripts across runs. After each episode, the full trajectory (observations, actions, rewards) is appended to a growing context buffer. When the buffer exceeds the model's context window, a FIFO eviction policy drops the oldest episodes. At each step, the agent receives this accumulated history alongside the current observation, and selects actions informed by past experience. Memory improves over Static by giving the agent access to concrete prior examples, but it does not distill or structure

that experience: the agent must extract useful patterns from raw transcripts, and as context grows, earlier episodes are lost.

Reflexion Shinn et al. (2023). Reflexion augments the agent with a verbal self-reflection step after each episode. A second LLM call (or a second pass of the same model) reads the completed episode trajectory and produces a short structured reflection—typically a paragraph summarizing what went wrong, what should be tried differently, and what strategies were effective. These reflections are prepended to the system prompt of subsequent episodes, giving the agent a compact, curated summary of its own prior attempts. Reflexion is able to identify and fix local mistakes (e.g., avoiding a specific trap or correcting a failed action sequence), but its reflections describe what happened, not what the agent has not yet tried—leaving the unexplored strategy space implicit and untracked.

EvoTest He et al. (2025). EvoTest frames agent improvement as an evolutionary optimization over the agent's configuration: its system prompt, in-context memory bank, and tool definitions are all treated as mutable components subject to iterative revision. After a batch of episodes, an LLM proposes mutations—additions, deletions, and edits—to each component. Mutations that improve performance are retained; others are discarded. This gradient-free evolution enables EvoTest to discover improvements that fixed reflection prompts may miss, including changes to tool behavior and memory structure. However, EvoTest does not maintain an explicit strategy map: its evolved components encode aggregate experience but provide no structured view of which strategies have been tried and which remain unexplored.

ACE Zhang et al. (2025). ACE (Agentic Context Engineering) treats the agent's context as an evolving playbook that accumulates and refines strategies through a modular grow-and-refine loop. After each episode, new strategies are extracted and appended to the playbook (the *grow* step); periodically, an LLM consolidates redundant or outdated entries (the *refine* step). The playbook is injected into the agent's context at each episode, providing a curated collection of reusable strategies. ACE differs from Reflexion in that it maintains a persistent, editable strategy bank rather than episode-level reflections, and from EvoTest in that its representation is human-readable and modular. Like both, however, ACE does not track which strategies remain unexplored: growth is driven by what the agent has encountered, not by what it has systematically avoided.

C Implementation Details

Hyperparameters. Table 10 summarizes the hyperparameters used in all main experiments unless otherwise stated.

Table 10: APEX hyperparameters used in main experiments.

Parameter	Value	Description
Reflect interval N	5	Episodes between reflection cycles
Discount factor γ	0.6	Return propagation discount
UCB constant c	10	Exploration bonus coefficient
ϵ	0.1	Exploration probability (ϵ-Greedy)
Max lessons	20	Global lesson buffer capacity
Fork Discovery ops	6	Max new milestones per reflection cycle
Exploration freeze	30	Episode at which Fork Discovery is disabled
Episodes	50	Per game (Jericho); 5 per task (WebArena)
Steps per episode	120	Environment step limit

Milestone Completion and Auto-Skip. At each step, the agent includes a `current_milestone_completed` flag in its structured output, by which it self-reports whether the current milestone has been achieved. The agent tracks how many steps it has spent on the current milestone; if this exceeds a patience threshold (40 steps for unvisited milestones with $n(v) = 0$, 20 for previously attempted ones) without the agent setting this flag, the milestone is automatically skipped and the agent advances to the next one. If a milestone is skipped, the agent does not retry it during the same episode. Skipping does not count as completion, so milestones requiring it remain unavailable.

Attempt Notes. Every N episodes, as part of the reflection cycle, the system extracts attempt notes from $\tilde{H}_t$ for each milestone in the recent episode paths. Each note captures whether the milestone was achieved (with reward and step number) or failed (with failure reason), and is stored in a per-milestone rolling buffer of the 5 most recent attempts. These notes serve as the input to Stuck Node Diagnosis.

Statistics During Deduplication. When Map Refinement prunes a duplicate node, the pruned node's bandit statistics θ_v and attempt notes are discarded; the surviving node retains its own accumulated statistics unchanged. This is justified because pruned nodes are typically lower-quality duplicates with inaccurate key actions or descriptions, making their statistics unrepresentative of the milestone's true value. Merging such statistics into the surviving node could corrupt its estimates.

Stuck Node Diagnosis. Every N episodes, the system identifies milestones that have been repeatedly attempted without success ($n(v) \geq 3$, $\bar{r}(v) \leq 0$), i.e., nodes that Policy Selection (Section 4.1) keeps targeting but that the agent consistently fails to achieve. A diagnostic LLM call examines the attempt note buffer and the node's dependency context to identify the root cause (e.g., a missing prerequisite not yet encoded in $\mathcal{G}$) and produce targeted guidance. This guidance is injected into the agent's prompt when the milestone is next attempted. To avoid redundant calls, each node has a cooldown of $2N$ episodes before it can be diagnosed again.

Global Lesson Extraction. Every N episodes, after Fork Discovery, an LLM distills cross-episode patterns from $\tilde{H}_t$ into reusable categorical lessons. Three categories are maintained: PENALTY lessons record actions that caused score loss, NAVIGATION lessons capture movement constraints and shortcuts, and MECHANIC lessons encode environment rules and object interactions. At most 5 new lessons are added per cycle. The full lesson buffer is injected into the system prompt at every step, giving the agent persistent cross-episode knowledge that complements the per-milestone guidance from the strategy map.

Policy Selection Parameters. All three selection strategies share the same treatment of unvisited milestones: nodes with $n(v) = 0$ receive $+\infty$ scores and are selected uniformly at random, ensuring full coverage before exploitation begins.

Thompson Sampling. For milestones visited exactly once, we sample from $\mathcal{N}(\bar{r}(v), \sigma_{\text{prior}})$ with $\sigma_{\text{prior}} = 100$ to maintain high initial uncertainty. For milestones with multiple visits, the standard error is used with a minimum floor of $\sigma_{\min} = 1.0$ to prevent premature convergence.

UCB. The exploration constant is set to $c = 10$, tuned to match the reward scale of the games (per-episode scores typically range from 0 to 50).

ϵ-Greedy. The exploration probability is set to $\epsilon = 0.1$.

Strategy Map Grouping for WebArena. Unlike Jericho, where each game has a dedicated strategy map, WebArena comprises many distinct tasks. We group tasks dynamically: when a new task arrives, an LLM judges whether it is sufficiently similar to any previously seen task; if so, the new task reuses that task's strategy map, otherwise a new map is created. This allows the strategy map to accumulate knowledge across related tasks without requiring predefined task categories. The cross-task evaluation in Table 3 uses experience from prior tasks, without previous attempts on the current task. It therefore tests transfer among tasks from the same distribution with reusable subgoals, rather than generalization to a new environment distribution.

Compute Resources. All experiments use API-based LLM inference with no local GPU required. A standard laptop suffices for orchestration; the only resource constraint is LLM API quota. No preliminary or failed experiments outside those reported were conducted at scale.

D ADDITIONAL RESULTS

D.1 RESULTS WITH STANDARD DEVIATION

Tables 11 and 12 report mean $\pm$ std for Jericho and WebArena respectively. All std values use sample standard deviation ($n - 1$). These values describe variation among the final K episodes of one run, not uncertainty across independent runs; separate three-run checks appear in Appendix G.

Table 11: Jericho Final-5 mean ± sample std across episodes 46–50 of one run (50 episodes, 120 steps each), not across independent runs.

Method	Zork1	Balances	Zork3	Temple	Pentari	Ludicorp	Deephome	Detective	Ztuu
Static	39.8±5.3	11.0±4.2	2.8±0.4	6.2±1.6	6.0±2.2	13.6±1.5	17.2±1.8	70.0±54.8	0.0±0.0
Memory	46.0±13.9	13.0±4.5	2.4±0.5	6.2±1.6	7.0±2.7	17.0±6.2	29.0±21.1	130.0±0.0	0.0±0.0
Reflexion	59.8±6.7	20.0±5.0	3.0±0.0	9.0±2.2	10.0±0.0	35.0±11.7	29.6±6.7	144.0±98.4	5.0±0.0
EvoTest	42.0±4.5	18.0±2.7	2.4±0.5	10.0±0.0	16.0±5.5	48.0±11.3	56.8±31.3	314.0±5.5	25.0±2.7
ACE	46.0±9.6	18.0±4.5	2.2±0.4	7.2±2.2	19.0±2.2	19.2±2.2	8.4±1.5	198.0±55.0	3.4±5.3
APEX (Ours)	**73.0±13.7**	**23.0±2.7**	**4.6±0.9**	**20.8±1.1**	**46.0±4.2**	**58.8±9.1**	**103.4±0.9**	**322.0±4.5**	**29.0±6.7**

Table 12: WebArena Final-3 mean ± sample std across the final three episodes of one run (5 episodes per task), not across independent runs.

Method	Shopping	Reddit	GitLab	CMS	Map	Overall
Static	34.7±3.0	45.7±0.4	36.3±0.5	44.8±0.8	56.3±3.4	42.3±0.2
Memory	39.1±1.7	53.8±0.9	53.9±1.8	53.2±2.2	55.2±2.5	50.5±0.7
Reflexion	34.1±2.1	36.2±2.0	50.8±0.7	46.7±0.0	54.2±2.0	44.3±0.8
EvoTest	38.9±1.7	50.9±2.7	54.4±0.0	53.8±0.8	52.9±3.2	49.9±1.2
ACE	35.9±2.2	52.5±1.3	51.6±1.2	52.4±1.4	55.7±0.7	48.6±1.4
APEX (Ours)	**42.9±2.0**	**68.2±0.8**	**60.2±0.6**	**55.1±1.9**	**57.6±1.3**	**55.9±0.5**

D.2 LLM Backbone Comparison

To evaluate whether APEX's performance depends on the backbone LLM, we test three models while keeping all other settings identical. As shown in Table 13, APEX consistently outperforms all baselines across all three backbones, confirming that its gains reflect the strategy map and Fork Discovery rather than any particular model's capability.

Table 13: LLM backbone comparison (Final-5).

Game	Static	Memory	Reflexion	EvoTest	ACE	APEX
Gemini 3 Flash						
Zork1	39.8	46.0	59.8	42.0	46.0	**73.0**
Pentari	6.0	7.0	10.0	16.0	19.0	**46.0**
Ztuu	0.0	0.0	5.0	25.0	3.4	**29.0**
DeepSeek-V3.2						
Zork1	30.0	15.8	37.8	50.8	43.3	**64.4**
Pentari	16.0	5.0	22.0	26.0	10.0	**43.0**
Ztuu	0.0	0.0	3.8	18.8	1.5	**29.0**
GPT-5-mini						
Zork1	35.0	36.2	39.0	43.6	37.4	**59.8**
Pentari	5.0	10.0	5.0	11.0	8.0	**42.0**
Ztuu	5.0	2.8	3.4	8.4	4.8	**19.2**

D.3 Full Learning Curves

The learning curves on nine Jericho games are shown in Figure 6.

D.4 Prompt Templates

This section presents the core prompt templates used in APEX. Dynamic content (episode data, map state, etc.) is injected at the positions marked with placeholders.

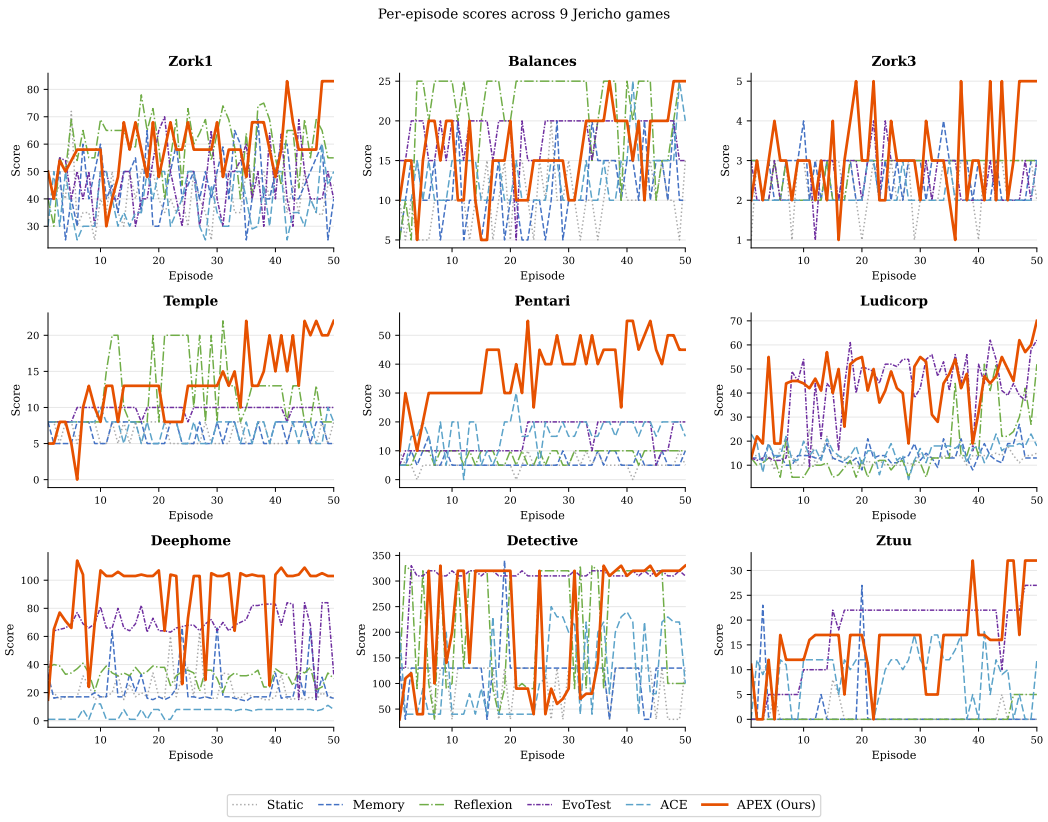


Figure 6: Per-episode scores across 50 episodes on all nine Jericho games.

D.5 Action Prompt

The agent receives a system prompt and a user prompt at each step. The system prompt is composed of a base instruction, global lessons, and strategy guidance derived from the current milestone.

Action Prompt — System Prompt

You are an expert agent aiming to complete an interactive task. Rewards are given for making progress. Select promising actions based on the current state and memory of past interactions.
LESSONS FROM PAST EXPERIENCE (follow these strictly):
- [PENALTY] *{lesson text}*
- [NAVIGATION] *{lesson text}*
- [MECHANIC] *{lesson text}*

Action Prompt — User Prompt

RECENT STEPS:
Step i: State: *{state}* Action: *{action}* Reward: *{reward}*
. . .
LAST 8 ACTIONS: $action_1 \rightarrow action_2 \rightarrow \ldots$
CURRENT STATE: *{current observation, agent state / inventory}*
PROGRESS: Step t/T (*remaining* steps left) | Reward: s
CURRENT MILESTONE: *{milestone description}*
  Key action sequence: *{key actions}*
  Known pitfalls: At [*{location}*]: agents often loop doing [*{pattern}*]. Instead, try: [*{escape action}*]
TASK:
1. Analyze your progress: What have you achieved? What's your next objective?
2. Propose your best next action with reasoning

RESPONSE FORMAT (JSON):
{"progress_analysis": "...", "current_milestone_completed": false, "next_objective": "...", "reasoning": "...", "action": "..."}
KEY RULES:
- If you haven't received rewards recently, you are likely stuck — try a fundamentally different approach
- LOOP DETECTION: Look at the LAST ACTIONS trail. If cycling between the same states, do something completely different

D.6 Episode Summary Prompt

After each episode, an LLM generates a milestone-centric structured summary used by all reflection modules. Milestones are significant task progress steps identified from the trajectory, regardless of whether they directly earned a reward.

Episode Summary — System Prompt

You are an expert at analyzing agent episode trajectories. Your job is to extract milestones — the key sub-goals that represent significant task progress — and analyze the path to each one.

Episode Summary — User Prompt

Final score: s, Total steps: T
STRATEGY MAP: *{strategy map}*
EPISODE TRAJECTORY:
Step i: STATE: *observation* | ACTION: *action* | REWARD: $+r$
. . .
Write the summary in this format:

- **Milestones Achieved**: matched to known nodes or marked [NEW] if unrecognized
- **Penalties Received**: actions that caused score loss
- **Milestones NOT Achieved**: with failure reason and missing prerequisites
- **Unexplored Opportunities**: actions available or observed but never tried (untried valid actions, unexamined objects, unexplored paths)

Critical: identify milestones by meaningful task progress steps, not by incidental state features.

D.7 Map Refinement Prompt

Map Refinement updates the strategy map based on observed facts from recent episodes. The LLM receives the current map state and episode summaries, then proposes structural operations.

Map Refinement — System Prompt

You are an expert at analyzing agent episode results and maintaining a strategy map.
The agent's knowledge is organized as a MILESTONE DAG (directed acyclic graph). Each node has a "deps" list meaning "this milestone REQUIRES these specific prerequisites to be completed first". deps=[] means the milestone is independent.
Your job is to UPDATE the DAG based on OBSERVED FACTS from the episodes below. You must ONLY encode things that actually happened — do NOT speculate.
RULES:
(1) Every significant event that **actually occurred** should correspond to a node — reward changes, state transitions, key resource acquisitions. Do NOT add nodes for things the agent never tried.
(2) Fix incorrect key_actions with update_node; do not rename nodes with confirmed rewards.
(3) Prune only to remove duplicate nodes.
(4) Only add a dependency when the milestone is impossible without the prerequisite.
(5) When updating an existing node, make only minor corrections (fix key_actions, adjust dependency edges). If the core goal of the milestone changes fundamentally, create a new node with a fresh identity instead.
(6) To correct a wrong dependency edge, use `update_deps` to move the node to a different parent. This preserves the node's statistics and subtree.

Map Refinement — User Prompt

STRATEGY MAP: *{strategy map}*
EPISODE SUMMARIES: *{structured summaries}*
Step 1 — Analyze each episode: list every significant event and check against DAG.
Step 2 — Analyze milestone failures: check for missing prerequisites.
Step 3 — Dedup check and dependency audit.
Step 4 — Propose DAG operations: add_child, add_branch, update_node, update_deps, or prune.

D.8 Fork Discovery Prompt

Fork Discovery identifies unexplored directions — actions the agent observed or had available but never pursued. It takes episode summaries (Section D.6) and the current map as input.

Fork Discovery — System Prompt

You are an expert at identifying unexplored opportunities in agent episode trajectories. Your task is to identify actions the agent observed or had available but never pursued, which could unlock new content.
CRITICAL CONSTRAINTS: Every proposed milestone must be a concrete, executable action grounded in evidence from the episodes. Limit to at most 6 new milestones per reflection.

Fork Discovery — User Prompt

STRATEGY MAP: *{strategy map}*
EPISODE TRAJECTORIES: *{episode summaries}*
Identify unexplored opportunities: find actions or directions that were available or observable in the episodes but never attempted. Propose new milestone nodes with appropriate dependency edges to encode these discoveries into the strategy map.

D.9 Stuck Node Diagnosis Prompt

Stuck Node Diagnosis analyzes milestones that persistently fail ($n \geq 3$, $\bar{r} \leq 0$) to produce targeted guidance.

Stuck Node Diagnosis — System Prompt

You are an expert at diagnosing why an agent keeps failing at certain milestones.
Analyze the stuck node's attempt notes and context to determine: (1) WHY the agent keeps failing (root cause), (2) What specific action the agent should take next time, (3) Whether the failure is due to a missing prerequisite or unmet dependency.

Stuck Node Diagnosis — User Prompt

Analyze this stuck node:
— *node_id*: *milestone* —
key_action: *{actions}*
visits: *n*, avg_reward: *r*
Recent attempt notes: *{last 5 attempts}*
Depends on: *{upstream nodes}*
Downstream: *{dependent nodes}*
Respond with: **Root cause** (why it keeps failing), **Next action** (what to try), **Missing prerequisite** (if applicable).

D.10 Global Lesson Extraction Prompt

Global Lesson Extraction distills cross-episode patterns into reusable categorical lessons.

Global Lesson Extraction — System Prompt

You are an expert at extracting reusable lessons from agent episode trajectories.

Extract GENERAL lessons the agent should remember across all future episodes. Three categories: PENALTY records actions that caused reward loss; NAVIGATION captures movement constraints and shortcuts; MECHANIC encodes environment mechanics and object interactions.
Accuracy rules: only record observed facts, require evidence for conditional discoveries. Maximum 5 new lessons per call.

Global Lesson Extraction — User Prompt

{Recent episode summaries}
EXISTING LESSONS: *{current lesson buffer}*
What general lessons should the agent remember for all future episodes?

E Asset Licenses

Table 14 lists the open-source assets used in this paper along with their licenses.

Table 14: Licenses for assets used in this paper.

Asset	Source	License
Jericho Hausknecht et al. (2020)	`https://github.com/microsoft/jericho`	MIT
WebArena Zhou et al. (2024)	`https://github.com/web-arena-x/webarena`	Apache 2.0
Reflexion Shinn et al. (2023)	`https://github.com/noahshinn/reflexion`	MIT
EvoTest He et al. (2025)	`https://github.com/microsoft/EvoAgent`	MIT
ACE Zhang et al. (2025)	`https://github.com/acewrench/ACE`	MIT

F Controlled Exploration Diagnostics

We extend the deterministic 5×5 maze diagnostic to seven backbones and five methods. Each method–backbone pair is evaluated over five seeds, with 20 episodes per seed and the same 10-step episode budget as the motivating maze. The controlled comparisons keep prompts, budget, decoding settings, and initial state fixed when changing the backbone; method-specific memory and exploration mechanisms remain part of each method. The +40 location provides an accessible rewarding route, while reaching the +80 room requires discovering an alternative.

Metrics. The +80 discovery rate is the fraction of the five seeds that reach the +80 room at least once during the run. It measures discovery, not the frequency of subsequently exploiting that route. For a seed, let $c(x)$ count visits to state x over the run and $p(x) = c(x)/\sum_y c(y)$. Exploration entropy is

$$H_{\text{visit}} = -\sum_{x:p(x)>0} p(x) \log_2 p(x). \tag{9}$$

This is entropy of cumulative state visitation, not token-level policy entropy or an estimate of entropy at each training episode. Table 16 reports the mean and standard deviation across seeds.

Table 15: Maze +80 discovery rate (% of five seeds; higher is better). Each seed runs 20 episodes. Model labels follow the experiment records.

Backbone	Reflexion	Memory	EvoTest	ACE	APEX
Claude Opus 5	20	0	0	20	**100**
Claude Opus 4.8	0	20	0	0	**100**
GPT-5.6-sol	40	0	20	40	**60**
GLM-5.2	20	0	0	0	**60**
DeepSeek-V4-Pro	0	0	0	0	**100**
DeepSeek-V4-Flash	20	0	0	0	**60**
Gemini 3.5 Flash	60	40	0	0	**80**

Table 16: Cumulative visitation entropy in bits (mean ± std over five seeds; higher is better).

Backbone	Reflexion	Memory	EvoTest	ACE	APEX
Claude Opus 5	$3.26 \pm .14$	$3.42 \pm .22$	$3.27 \pm .16$	$3.18 \pm .18$	$\mathbf{4.17 \pm .14}$
Claude Opus 4.8	$3.26 \pm .19$	$4.03 \pm .11$	$3.27 \pm .06$	$2.86 \pm .36$	$\mathbf{4.08 \pm .31}$
GPT-5.6-sol	$3.51 \pm .38$	$3.31 \pm .11$	$3.57 \pm .15$	$3.25 \pm .46$	$\mathbf{3.99 \pm .30}$
GLM-5.2	$3.81 \pm .20$	$3.78 \pm .25$	$3.16 \pm .26$	$2.37 \pm .20$	$\mathbf{4.08 \pm .13}$
DeepSeek-V4-Pro	$3.78 \pm .19$	$3.53 \pm .15$	$2.84 \pm .19$	$1.95 \pm .32$	$\mathbf{4.34 \pm .15}$
DeepSeek-V4-Flash	$3.63 \pm .37$	$3.53 \pm .09$	$3.03 \pm .10$	$2.19 \pm .14$	$\mathbf{4.10 \pm .18}$
Gemini 3.5 Flash	$3.94 \pm .36$	$4.00 \pm .20$	$3.38 \pm .08$	$3.18 \pm .31$	$\mathbf{4.19 \pm .20}$

APEX has the highest discovery rate and mean visitation entropy for every tested backbone. However, broad visitation and discovery are distinct: on Claude Opus 4.8, Memory has entropy 4.03 ± 0.11, close to APEX's 4.08 ± 0.31, but discovers +80 in only one of five seeds. The results support measuring discovery alongside diversity. Five seeds provide limited precision, and these aggregates alone do not establish that entropy decreases over time.

G Independent-Run Results

We repeat all six main methods with three independent seeds on Zork1, Pentari, Ztuu, Reddit, and GitLab, using the main Gemini 3 Flash backbone. For each seed we first compute its Final-K score ($K = 5$ for Jericho and $K = 3$ for WebArena), then report the sample mean and standard deviation across the three run-level values:

$$\bar{z} = \frac{1}{3}\sum_{j=1}^{3} z_j, \qquad s_z = \sqrt{\frac{1}{2}\sum_{j=1}^{3}(z_j - \bar{z})^2}. \tag{10}$$

These statistics measure run-to-run variability, unlike the late-episode standard deviations in Tables 11 and 12. APEX has the highest mean in each checked setting. This independent-run evaluation covers three Jericho games and two WebArena domains, rather than the full benchmarks.

Table 17: Three independent runs: mean ± sample std of run-level Final-K scores. WebArena scores are percentages.

Method	Zork1	Pentari	Ztuu	Reddit	GitLab
Static	37.4 ± 3.6	8.0 ± 2.6	0.0 ± 0.0	45.2 ± 1.4	36.0 ± 0.9
Memory	48.2 ± 2.9	4.7 ± 1.5	2.2 ± 2.1	53.1 ± 1.7	53.6 ± 1.3
Reflexion	57.6 ± 3.6	12.3 ± 2.3	3.0 ± 1.4	36.5 ± 2.4	49.3 ± 0.9
EvoTest	44.3 ± 3.7	13.7 ± 2.3	27.2 ± 2.1	50.3 ± 2.6	54.1 ± 0.7
ACE	43.8 ± 4.4	21.3 ± 3.1	1.0 ± 1.0	52.0 ± 1.6	52.4 ± 1.2
APEX	$\mathbf{75.1 \pm 3.7}$	$\mathbf{43.7 \pm 3.1}$	$\mathbf{31.0 \pm 1.7}$	$\mathbf{67.4 \pm 1.8}$	$\mathbf{59.7 \pm 1.1}$

H WebArena Paired Comparisons

Tables 18 and 19 report APEX minus baseline in percentage points and the corresponding two-sided paired t-test p-value, using task instances as paired units. The Overall test pools task instances across domains; it is not a test on five domain means or on the three independent runs above. These are per-comparison tests; no multiple-comparison-adjusted conclusions are claimed.

The reported Overall comparison has $p < .05$ against each baseline in both settings. This does not imply that every domain improves significantly. Against Memory, only Reddit and GitLab reach that threshold in the main setting; only Reddit does so in the cross-task setting. The cross-task CMS difference is negative. Pooled results weight tasks rather than assigning equal weight to each domain, which explains why the Overall difference need not equal the mean of the five displayed differences.

Table 18: Main WebArena results: for each baseline, the first row is the difference in percentage points and the second row gives reported paired-test p-values.

Baseline	Shopping	Reddit	GitLab	CMS	Map	Overall
Static	+8.2	+22.5	+23.9	+10.3	+1.3	+13.6
p	.022	$< .001$	$< .001$	.005	.587	$< .001$
Memory	+3.8	+14.4	+6.3	+1.9	+2.4	+5.4
p	.068	$< .001$	.018	.322	.236	$< .001$
Reflexion	+8.8	+32.0	+9.4	+8.4	+3.4	+11.6
p	.006	$< .001$	$< .001$	.017	.079	$< .001$
EvoTest	+4.0	+17.3	+5.8	+1.3	+4.7	+6.0
p	.059	.005	.018	.363	.111	.006
ACE	+7.0	+15.7	+8.6	+2.7	+1.9	+7.3
p	.015	$< .001$	.006	.125	.110	.007

Table 19: Cross-task WebArena results: score difference and reported paired-test p-value, with the same conventions as Table 18.

Baseline	Shopping	Reddit	GitLab	CMS	Map	Overall
Static	+7.8	+12.1	+20.0	+8.8	+6.8	+11.1
p	.034	.002	$< .001$	.014	.060	$< .001$
Memory	+4.2	+3.9	+2.7	−0.4	+0.9	+2.3
p	.052	.043	.110	.824	.619	.024
Reflexion	+8.9	+21.7	+6.4	+6.5	+7.4	+10.2
p	.014	$< .001$	.021	.041	.028	$< .001$
EvoTest	+4.2	+8.9	+2.5	+0.0	+5.6	+4.2
p	.042	.031	.041	1.000	.078	.004
ACE	+15.1	+10.1	+5.4	+3.8	+4.6	+7.8
p	$< .001$	.004	.032	.054	.049	$< .001$

I Additional Baselines and Component Ablations

Retrievable skill bank. To test whether a simpler memory structure explains the gain, we construct a test-time skill bank using the Reflexion loop. Each episode produces one named skill. Skills are deduplicated by exact title and embedding similarity, and the top five relevant skills are retrieved for the current state and injected into the action prompt. This baseline evaluates test-time memory retrieval without supervised fine-tuning, reinforcement-learning weight updates, DAG dependencies, node-level bandit statistics, or Fork Discovery.

Table 20: Retrievable skill memory and auxiliary-component checks. Final-5 on Jericho under the 50-episode protocol; values are point estimates rather than three-run means.

Method / configuration	Zork1	Pentari	Ztuu
Reflexion	59.8	10.0	5.0
Retrievable skill bank	66.8	9.0	8.2
APEX without Global Lesson Extraction	66.0	38.0	24.2
APEX without Stuck Node Diagnosis	64.0	34.0	18.2
APEX (full)	**73.0**	**46.0**	**29.0**

APEX exceeds the skill bank on all three games. Removing either auxiliary component decreases the full system's point estimate, showing that both contribute under these settings. These individual removals do not establish additive or independent effects; no joint-removal result is reported here.

Reward attribution. The LLM maps observed score gains to milestones. An audit over one 50-episode run for each of the nine Jericho games reports 92.0% reward capture and 99.4% agreement between assigned rewards and environment rewards. These descriptive audit rates concern Jericho and do not establish attribution accuracy in sparse-reward WebArena. In WebArena, the LLM additionally supplies milestone-level feedback.

Table 21: Reward-attribution comparison on Jericho (Final-5). The environment-only variant splits observed score gains across attempted milestones.

Attribution variant	Zork1	Pentari	Ztuu
Environment rewards only	71.2	45.0	28.2
LLM attribution (APEX)	**73.0**	**46.0**	**29.0**

The environment-only variant is within 1.8, 1.0, and 0.8 points of the full system on the three games. This suggests that the strategy map remains useful without LLM attribution on these Jericho tasks. These point estimates do not establish a statistically significant advantage for LLM attribution.

J Strategy-Map Size and Node Lifecycle

Measured map size. Table 22 summarizes one complete 50-episode run per Jericho game. The columns distinguish total, active, and pruned nodes as recorded in the map-size audit. Final active counts range from 16 to 57, and estimated serialized-map sizes range from 0.5k to 1.9k tokens. These are estimates for the map representation, not the full prompt, total inference tokens, or measured latency.

Table 22: Final map-size audit from nine Jericho runs. Token counts are approximate serialization estimates. Pruned counts use the map-size audit's accounting and should not be equated with the lifecycle categories in Table 23.

Game	Total nodes	Active	Pruned	Dep. edges	Est. tokens
Zork1	57	57	0	66	1.9k
Balances	52	39	13	31	1.4k
Zork3	42	38	4	37	1.3k
Temple	33	31	2	27	1.0k
Pentari	27	16	11	12	0.5k
Ludicorp	58	55	3	58	1.8k
Deephome	54	54	0	70	1.8k
Detective	35	34	1	34	1.1k
Ztuu	30	28	2	17	0.9k

The recorded growth slows around the episode-30 Fork Discovery freeze, and some games show later reductions. The fixed proposal cap, freeze, and pruning contribute to this behavior, so it is not evidence of a task-complexity-only bound or a large-map stress test.

J.1 Proposal Lifecycle and Retirement

New milestones enter with $n(v) = 0$ and receive priority once eligible. Their dependencies can still prevent an immediate attempt. The reported retirement criterion targets nodes with at least three attempted episodes, non-positive mean return, and no rewarding descendants. Such nodes are excluded from future selection. Diagnostic guidance and dependency edits address repeated execution failures; deduplication removes redundant representations. These mechanisms reduce continued allocation to unproductive proposals but do not establish a guarantee of semantic correctness or exploration completeness.

For retired nodes, the reported median delay from creation to retirement is five episodes. Retirement groups together incorrect and merely low-value proposals; 27.3% must not be interpreted as a hallucination rate. Likewise, execution validation is not an independent audit of every dependency edge.

Parameters and operating scope. Episode and step budgets specify evaluation resources. The UCB coefficient and ϵ are inactive in the default Thompson-style variant. Reflection interval, return discount, proposal/lesson caps, exploration freeze, patience, and uncertainty priors remain method settings even when fixed across tasks. In particular, $\sigma_{\text{prior}} = 100$ and $\sigma_{\min} = 1$ mean that the implemented Gaussian sampler is not parameter-free.

Table 23: Reported lifecycle of 389 LLM-proposed nodes across nine 50-episode Jericho runs. Percentages are rounded; categories are outcomes in the lifecycle audit.

Recorded outcome	Proportion
Validated by execution	58.6%
Merged/pruned as duplicates	10.0%
Retired by statistics	27.3%
Still under evaluation	4.1%

K Additional Backbone Results

The following checks supplement Appendix D.2. They change the backbone while keeping the reported prompts, episode budgets, decoding, and method configurations fixed. Tables report Final-K point estimates, not the three-run statistics in Appendix G. Coverage is partial: the WebArena checks compare only Memory, EvoTest, and APEX on two domains.

Table 24: Claude Opus 4.8 on three Jericho games (Final-5).

Game	Static	Memory	Reflexion	EvoTest	ACE	APEX
Zork1	40.0	48.4	64.2	54.8	44.6	**81.6**
Pentari	6.0	8.0	10.0	16.0	18.0	**54.0**
Ztuu	0.0	0.0	5.0	25.8	3.4	**32.0**

Table 25: Additional Jericho backbone checks (Final-5). Each model block reports Deephome and Detective.

Game	Static	Memory	Reflexion	EvoTest	ACE	APEX
			Gemini 3 Flash			
Deephome	17.2	29.0	29.6	56.8	8.4	**103.4**
Detective	70.0	130.0	144.0	314.0	198.0	**322.0**
			DeepSeek-V3.2			
Deephome	18.4	30.6	34.2	61.8	16.8	**106.2**
Detective	62.0	116.0	138.0	90.0	176.0	**318.0**
			DeepSeek-V4-Pro			
Deephome	16.6	31.8	35.0	64.0	7.6	**111.0**
Detective	72.0	132.0	148.0	270.0	202.0	**330.0**
			GPT-5-mini			
Deephome	15.8	27.2	31.4	15.4	11.6	**101.8**
Detective	58.0	122.0	148.0	52.0	188.0	**312.0**

Table 26: Additional WebArena backbone checks (Final-3, %).

Backbone	Domain	Memory	EvoTest	APEX
GPT-5-mini	Reddit	52.7	50.9	**63.4**
GPT-5-mini	Shopping	38.4	37.2	**42.1**
DeepSeek-V4-Pro	Reddit	52.9	50.2	**69.4**
DeepSeek-V4-Pro	Shopping	38.6	38.1	**42.4**

L A Conditional Model of Experience Reinforcement

We give an explanatory abstraction, not an exact model of any particular LLM's memory or reflection update. Let $\mathcal{S}_{\mathrm{strat}}$ be a finite strategy set and π_t a distribution over it. Suppose experience

reinforcement follows

$$\pi_{t+1}(s) = \frac{\pi_t(s)\exp(\eta \widehat{R}_t(s))}{\sum_{u \in \mathcal{S}_{\text{strat}}} \pi_t(u)\exp(\eta \widehat{R}_t(u))}, \qquad \eta > 0. \tag{11}$$

Here $\widehat{R}_t$ is the value attributed from available experience; it need not equal a strategy's true return.

Conditional relative suppression. Assume $\pi_0(s_0) > 0$ and $\pi_0(s^\star) > 0$. If an early familiar strategy s_0 is favored in estimated value over an insufficiently explored $s^\star$ by at least $\Delta > 0$ at every update $k = 0, \ldots, t-1$, then

$$\frac{\pi_t(s^\star)}{\pi_t(s_0)} = \frac{\pi_0(s^\star)}{\pi_0(s_0)} \exp\left(\eta \sum_{k=0}^{t-1} [\widehat{R}_k(s^\star) - \widehat{R}_k(s_0)]\right) \leq \frac{\pi_0(s^\star)}{\pi_0(s_0)} e^{-\eta \Delta t}. \tag{12}$$

The equality follows by taking the ratio of the two updates, cancelling each normalizer, and multiplying across iterations. Applying the assumed gap gives the inequality. Thus, a strategy with higher true return can lose relative probability while the estimated gap persists.

This statement does not prove monotonic entropy decay, reduced visitation coverage, or shrinking mathematical support. It also does not show that every memory-based agent obeys the assumed update or maintains the estimated gap. The visitation and discovery measurements above provide separate empirical evidence.

Relation to APEX. Define $U_t = \{v \in \mathcal{A}(s, b) : n_t(v) = 0\}$. When U_t is nonempty, APEX chooses among these eligible unvisited candidates before exploiting visited ones. Fork Discovery can add candidates while enabled, and Map Refinement can revise their dependencies. This makes opportunities for explicit testing available, subject to eligibility, finite episode budgets, the freeze schedule, and the quality of the proposed frontier. It provides a mechanism for counteracting suppression of alternatives, not a guarantee of finding the globally optimal strategy.