RETRACE: FAST ADAPTATION VIA SLOW CONTEXTUAL META-LEARNING FOR WEB TRAVERSAL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Humans rarely solve search problems by persisting along a single path. When progress stalls, they adjust their search, reflect on what failed, and recover from unsuccessful attempts. They also retain what successful recovery teaches them and reuse it later. Web traversal agents often struggle to adapt when search progress stalls, while lessons from failure and recovery are rarely consolidated into reusable guidance. We introduce ReTrace, a two-timescale contextual meta-learning framework for web traversal agents that couples fast metacognitive adaptation with slow experience consolidation. At the fast timescale, ReTrace treats how to adapt search behavior as a meta-decision: the agent coordinates traversal, recovery, and reflection to navigate the web, recover from unsuccessful attempts, and reflect on failures to adapt within a task. These adaptations yield informative traces of search, recovery, and reflection. At the slow timescale, a meta-learning harness uses these traces together with task-level ground-truth feedback to attribute successful behavioral changes, consolidate them into three reusable skills—Traversal Skill, Recovery Skill, and Reflection Skill—and organize these skills into a contextual scaffold for future tasks. In ReTrace, the fast agent produces adaptive traces, the slow harness turns these traces into context, and the evolved context improves subsequent fast adaptation. On WebWalkerQA with matched search budgets, fast adaptation raises pass@4 from 47.17% to 65.09%. Adding slow contextual meta-learning further improves V-GEMS QA pass@1 from 47.14% to 55.00% and WebWalkerQA pass@4 from 65.09% to 66.98%, suggesting that learned experience improves both the initial search policy and subsequent fast adaptation.

1 INTRODUCTION

Web agents extend language models from reasoning over static context to actively acquiring information from real-world websites Qi et al. (2025); Chae et al. (2025); He et al. (2025). In web traversal tasks, the information required to answer a query is distributed across multiple pages connected through hierarchical hyperlinks, requiring the agent to navigate the site, locate answer-bearing pages, gather evidence, and synthesize a final response Wu et al. (2025a); Tong et al. (2026). Unlike reasoning over a fixed document, success therefore depends not only on understanding what has already been observed, but also on a sequence of navigation decisions that determines which information becomes available for reasoning in the first place.

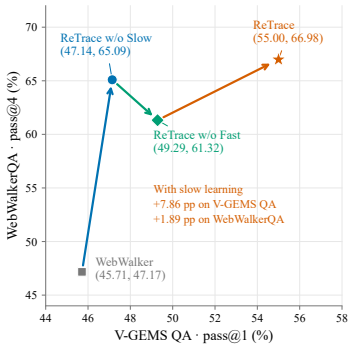


Figure 1: Fast/slow adaptation are complementary.

Existing web traversal agents typically navigate through a sequence of local decisions, selecting the next link from the current page and continually extending the search trajectory Tong et al. (2026). However, this local decision paradigm can become brittle when an early choice leads the agent into a wrong branch. Since web traversal is inherently hierarchical and branching, each link selection commits the search to a local direction, causing

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.

early mistakes to propagate through subsequent exploration and waste the limited interaction budget without bringing the agent closer to the required evidence. Humans, however, rarely persist along a single path when facing such complex search problems: when progress stalls, they reconsider the current direction, return to an earlier decision point, reflect on what went wrong, and explore an alternative. The hierarchical and branching structure of websites makes such adaptation particularly natural for web traversal, suggesting that an agent should reason not only about its next action, but also about the evolving search trajectory itself. This motivates a higher-level search control process that enables the agent to monitor and revise its trajectory online, supporting fast adaptation as the task unfolds.

Yet fast adaptation remains inherently transient if every new task requires the agent to rediscover the same correction through fresh trial and error Xiong et al. (2026). Simply preserving or retrieving past trajectories does not solve this problem: a trajectory records what happened in a particular search, but does not by itself reveal what should transfer to future tasks. Importantly, however, adaptation leaves behind something as valuable as the recovered answer itself: structured evidence about what failed, what changed, and what eventually worked. What is needed, therefore, is a slower learning process that can abstract reusable lessons from this evidence and consolidate them into contextual guidance for subsequent fast adaptation. The central question at this slower timescale is therefore not what to remember, but what to learn from how behavior changed. This mirrors how humans learn from recovery Wang et al. (2025): rather than remembering the exact path by which they escaped a failed search, they abstract the lesson behind that recovery and use it to approach later problems differently. This naturally leads to a contextual meta-learning view Mao et al. (2025): fast adaptation generates informative traces of failure and correction, while slow learning consolidates the transferable structure in those traces into the context that shapes future adaptation.

Based on this view, we introduce ReTrace, a two-timescale contextual meta-learning framework for web traversal agents. At the fast timescale, it performs metacognitive adaptation over the search trajectory. Beyond choosing the next link, it assesses whether to continue the current direction based on evidence, unresolved needs, progress, and budget. When progress stalls, it retreats and reroutes, rolling back both the navigation state and active context while retaining the failed direction only through reflection. These recovery transitions form structured meta-trajectories of behavioral adaptation. Task-level outcomes ground comparative reflection across attempts, without supervising individual recovery steps. At the slow timescale, a contextual meta-learning harness uses task-level ground-truth feedback to compare these meta-trajectories and consolidate recurring lessons into traversal, recovery and reflection skill. These skills are progressively organized into a contextual scaffold for subsequent tasks.

On WebWalkerQA with matched search budgets, fast adaptation raises pass@4 from 47.17% to 65.09%. Adding slow contextual meta-learning further improves V-GEMS QA pass@1 from 47.14% to 55.00% and WebWalkerQA pass@4 from 65.09% to 66.98%, suggesting that learned experience improves both the initial search policy and subsequent fast adaptation. ReTrace thus realizes fast adaptation via slow contextual meta-learning: the fast process learns how to recover within a task, while the slow process learns from those recoveries to shape more effective fast adaptation in future tasks. Our contributions are threefold:

- we formulate web traversal as fast metacognitive adaptation over search trajectories, enabling agents to retreat, reroute, and reflect when search stalls;
- we introduce a slow contextual meta-learning harness that uses task-level feedback and trajectory comparison to consolidate reusable skills into a contextual scaffold;
- we show that the two time scales in ReTrace are complementary, improving both repeated-attempt and first-attempt performance.

2 Related Work

Recent work has advanced web agents and their evaluation through stronger navigation, planning, multimodal grounding, and realistic web environments Deng et al. (2023); Zhou et al. (2024); Koh et al. (2024); He et al. (2024); Zheng et al. (2024); Lai et al. (2024). WebWalker introduces WebWalkerQA for hierarchical website traversal and an explore–critic framework for exploration and evidence assessment Wu et al. (2025a). Chae et al. augment web agents with world models that pre-

dict action consequences before policy selection Chae et al. (2025); Gu et al. (2025). WebExperT decomposes complex instructions with experiential fast-and-slow planning and refines decisions through failure reflection Luo et al. (2025). Related approaches improve web interaction through compositional policies and streamlined observation–action spaces Sodhi et al. (2023); Yang et al. (2025). Mango builds a global view of website structure, allocates budget across candidate entry points, and stores navigation history in episodic memory Tong et al. (2026). These methods improve where an agent starts, which action it selects, or how it plans ahead. However, they do not center whether the current branch remains worth pursuing: continuation versus abandonment is not treated as an explicit metacognitive decision grounded in evidence progress, unresolved needs, and remaining budget.

A parallel line of work learns from interaction experience, reflection, and memory Zhao et al. (2024); Shinn et al. (2023); Madaan et al. (2023); Park et al. (2023); Gupta et al. (2024). R2D2 combines a replay buffer with reflective memory to reconstruct visited web structure and refine strategies after errors Huang et al. (2025). OpenWebVoyager iterates real-world exploration, trajectory feedback, and policy optimization, learning from well-performing trajectories He et al. (2025). WebRL turns unsuccessful attempts into a self-evolving curriculum and trains agents with outcome-supervised reinforcement learning Qi et al. (2025). Agent Workflow Memory induces reusable workflows from past trajectories and selectively provides them for later web tasks Wang et al. (2025). Beyond web agents, memory-oriented systems show that experience can be stored, reflected upon, or abstracted into reusable behavior Zhong et al. (2024); Packer et al. (2023); Wang et al. (2024). Yet reuse can also propagate mistakes: Xiong et al. identify experience-following behavior, including error propagation and misaligned replay Xiong et al. (2026). More fundamentally, storage or retrieval alone does not reveal which change between a failed attempt and a corrected one should transfer. This motivates learning from contrasts across adaptive traces rather than treating a stored trajectory or retrieved episode as the transferable unit.

Our slow process is also related to contextual and meta-learning views of adaptation Akyürek et al. (2023); von Oswald et al. (2023); Garg et al. (2022). Mao et al. distinguish in-context skill recognition from skill learning, emphasizing that context can support adaptation beyond recalling a known pattern Mao et al. (2025). Wu et al. interpret in-context learning from a learning-to-learn perspective and show that transformers can construct data-dependent learning algorithms from context Wu et al. (2025b). A recent survey of agent memory similarly describes a progression from trajectory storage, through reflection, to experience abstraction, and highlights cross-trajectory abstraction for continual improvement Luo et al. (2026). ReTrace connects these ideas directly to web traversal: fast metacognitive recovery generates structured failure–correction traces, while a slow harness uses task-level outcomes and cross-attempt comparison to consolidate reusable traversal, recovery, and reflection skills into an evolving contextual scaffold. Rather than treating fast recovery and long-term experience learning as separate capabilities, ReTrace closes the loop between them: what is discovered during recovery becomes context that shapes future adaptation.

3 RETRACE: RECOVER TO LEARN, LEARN TO RECOVER

ReTrace, following the principle of recovering traces to learn and learning to recover traces, is a dual-timescale contextual meta-learning framework for within-task adaptation and cross-task learning. At the fast timescale, the agent responds to unproductive exploration by revising both its admissible navigation actions and its active reasoning context. At the slow timescale, a contextual meta-learning harness reviews adaptive traces from completed tasks and consolidates recurring lessons into three reusable skills: Traversal Skill, Recovery Skill, and Reflection Skill. These skills form a contextual scaffold that guides subsequent tasks without updating the model parameters.

The two timescales use failed experience in different ways. During execution, Fast removes an abandoned suffix from the active context, blocks the navigation edge that led into it, and retains a compact reflection to guide further search. The task record preserves the removed experience alongside the behavior that follows recovery. The slow harness draws on these records to identify recurring solving strategies and update the persistent contextual scaffold. Fast thus supplies the adaptation evidence for Slow, whose updates guide exploration and adaptation on future tasks.

For task q_i, the executor starts from a root page with a finite interaction budget. Each decision depends on the current navigation state x, active execution context m, remaining budget b, and textual

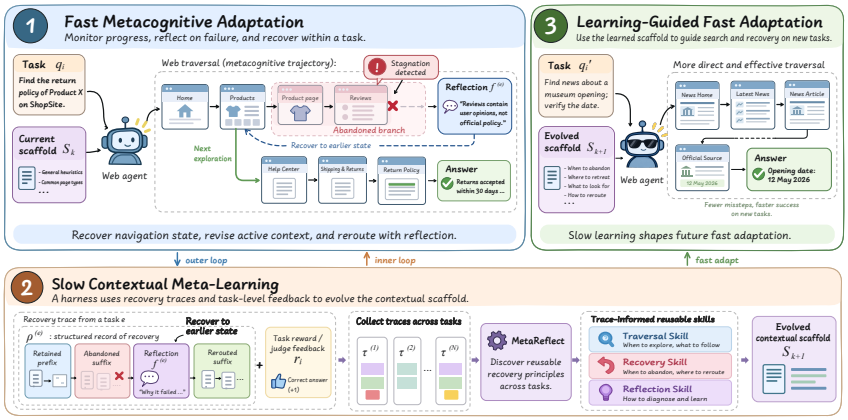


Figure 2: Overview of ReTrace. Recovery traces collected during fast adaptation are meta-learned into reusable traversal, recovery, and reflection skills, which guide more direct and effective behavior on subsequent tasks.

contextual scaffold S_k. The scaffold is fixed throughout learning window k. Fast adaptation changes only task-local actions and context; the slow harness updates S_k between windows. Reference answers and evaluator feedback become available only after task execution ends.

Fast adaptation responds when further exploration along the current trajectory becomes unproductive. A complete run consists of E_i execution episodes separated by recovery events. Each episode e has a logical trajectory $\tau_i^{(e)}$. After recovery, episode $e+1$ reuses an executed prefix and generates a new suffix without executing the prefix again. These episodes belong to a single run and are distinct from the complete runs used to compute pass@k.

At each step, Fast assesses whether the current search direction remains productive:

$$v_{i,t}^{(e)} = V\left(q_i, x_{i,t}^{(e)}, m_{i,t}^{(e)}, b_{i,t}^{(e)}\right), \qquad v_{i,t}^{(e)} \in \{0, \ldots, 10\}. \tag{1}$$

The assessment considers the evidence collected, unresolved information needs, recent progress, and remaining action and recovery budgets. Recovery is triggered when $v_{i,t}^{(e)} < \delta$, provided that a recovery point is available and enough recovery budget remains. The same underlying language model produces this score using a separate fixed prompt; no separately trained evaluator is used. The triggering mechanism stays fixed throughout slow contextual meta-learning.

When recovery is triggered, Fast uses a planner with a fixed prompt to select a recovery point $d_i^{(e)}$ and generate a task-local reflection $f_i^{(e)}$:

$$\left(d_i^{(e)}, f_i^{(e)}\right) = P\left(q_i, \mathcal{C}_i^{(e)}, \tau_i^{(e)}\right), \tag{2}$$

where $\mathcal{C}_i^{(e)}$ contains the available recovery points and their associated outgoing navigation edges. The planner selects a point whose outgoing edge should no longer be pursued. Its reflection explains why that direction was unproductive and how to revise the search. Both outputs rely solely on task-local experience. The continuation assessment decides whether to recover; the planner decides where to resume and how to adjust the search.

Let $\mathcal{Z}_i^{(e)}$ denote the set of blocked navigation edges, $g_i^{(e)}$ the outgoing edge selected for blocking, and let $d_i^{(e)}$ index its source state. Recovery initializes the next episode from the state and context immediately before that edge:

$$\begin{aligned} \mathcal{Z}_i^{(e+1)} &= \mathcal{Z}_i^{(e)} \cup \{g_i^{(e)}\}, \\ x_{i,0}^{(e+1)} &= x_{i,d_i^{(e)}}^{(e)}, \\ m_{i,0}^{(e+1)} &= \mathrm{Update}\left(m_{i,d_i^{(e)}}^{(e)}, f_i^{(e)}\right). \end{aligned} \tag{3}$$

The selected edge stays blocked for the rest of the run, preventing an immediate return along that edge. Restoring the navigation state sets the point from which execution resumes. Revising the active context removes observations and decisions introduced after that point, while the reflection retains a compact lesson from the abandoned search. Edge blocking restricts the available actions, and context revision changes the reasoning state used to choose among them. Recovery does not replenish the interaction budget, which continues to account for actual execution.

The executor then resumes under the same contextual scaffold:

$$\begin{aligned} a_{i,t}^{(e+1)} &\sim \pi_\theta\Big(\cdot \mid q_i, x_{i,t}^{(e+1)}, m_{i,t}^{(e+1)}, b_{i,t}^{(e+1)}; S_k\Big), \\ a_{i,t}^{(e+1)} &\in \mathcal{A}\Big(x_{i,t}^{(e+1)}; \mathcal{Z}_i^{(e+1)}\Big), \end{aligned} \tag{4}$$

where $\mathcal{A}(x; \mathcal{Z})$ is the set of admissible actions after blocked navigation edges are excluded. The resulting logical trajectory is

$$\tau_i^{(e+1)} = \tau^{(e)}_{i,\le d_i^{(e)}} \oplus \widetilde{\tau}_i^{(e+1)}, \tag{5}$$

where $\widetilde{\tau}_i^{(e+1)}$ is the newly executed suffix. The retained prefix preserves earlier progress, and the new suffix is generated under the revised action set and context.

The abandoned suffix is removed from active context but retained for later learning. Each recovery is recorded as a recovery trace:

$$\rho_i^{(e)} = \left(\tau^{(e)}_{i,\le d_i^{(e)}}, \tau^{(e)}_{i,> d_i^{(e)}}, f_i^{(e)}, \widetilde{\tau}_i^{(e+1)}\right). \tag{6}$$

This recovery trace links the shared execution prefix and abandoned continuation to the reflection and subsequent behavior. It preserves both the local change in search behavior and the experience that prompted it, whether or not the revised search ultimately succeeds.

At the slow timescale, the contextual meta-learning harness extracts recurring lessons from completed-task traces and incorporates them into the contextual scaffold. For a completed run, the full adaptation history is

$$\mathcal{H}_i = \left(\rho_i^{(1)}, \ldots, \rho_i^{(E_i-1)}, \tau_i^{(E_i)}\right). \tag{7}$$

If no recovery occurs, $\mathcal{H}_i$ contains only the final trajectory. When a task has multiple complete runs, each run's adaptation history and outcome are retained separately.

After execution ends, task-level feedback is obtained as

$$r_i = \mathrm{Judge}(q_i, \hat{y}_i, y_i), \tag{8}$$

where $\hat{y}_i$ is the submitted answer and y_i is the reference answer. This feedback evaluates the complete run; it does not label individual recovery decisions as correct or incorrect.

At each learning-window boundary, the slow harness collects the completed records

$$\mathcal{B}_k = \{(q_i, \mathcal{H}_i, \hat{y}_i, y_i, r_i)\}_{i \in \mathcal{I}_k}, \tag{9}$$

where $\mathcal{I}_k$ indexes the records completed in window k, and

$$\mathcal{B}_{\le k} = \bigcup_{j \le k} \mathcal{B}_j$$

denotes all experience accumulated through that window. A persistent analysis ledger retains previously identified evidence and candidate lessons across windows.

The harness examines both successful and unsuccessful searches for recurring patterns. It considers which information needs remained unresolved, which exploration choices failed to address them, how behavior changed after recovery, and whether the revised search eventually yielded a correct answer. Each recovery trace preserves the abandoned and revised continuations from a shared prefix, providing a structured comparison of search behavior before and after recovery. These comparisons support strategy induction, but they do not provide independent supervision or causal validation for individual recovery decisions.

The scaffold update has separate stages for evidence analysis, synthesis, and review. Evidence analysis identifies candidate lessons supported by completed-task records and checks whether the current scaffold already covers them. Supported lessons are then expressed as reusable conditional strategies that specify an observable situation, an actionable response, and an appropriate stopping condition, and are organized into the Traversal Skill, Recovery Skill, and Reflection Skill. The published contextual scaffold excludes task-specific identities, URLs, answers, and case details to retain general solving principles without memorizing individual instances. Before incorporation, a final review checks each candidate against the existing scaffold and its supporting evidence.

Formally, the update is

$$\begin{aligned} \mathcal{P}_k &= \mathrm{MetaReflect}(S_k, \mathcal{B}_{\le k}, \mathcal{L}_k)\,, \\ S_{k+1} &= \mathcal{U}(S_k, \mathcal{P}_k), \qquad \ell(S_{k+1}) \le B_S, \end{aligned} \tag{10}$$

where $\mathcal{L}_k$ is the persistent analysis ledger, $\ell(\cdot)$ measures the length of the published contextual scaffold, and B_S is a fixed scaffold length budget. If no candidate revision is approved, $S_{k+1} = S_k$. The ledger persists across windows and is not exposed to the executor.

The updated contextual scaffold applies only to subsequent tasks. Its strategies for evidence acquisition, complementary exploration, verification, and stopping guide both initial search and behavior after recovery. Following a recovery, the scaffold helps the executor interpret the task-local reflection and choose among the remaining admissible actions in Equation 4. Only the contextual scaffold changes across learning windows. The continuation assessment and recovery planner remain fixed, so Slow influences future Fast adaptation through executor behavior, without directly changing when or where recovery is triggered.

Fast and Slow learn from the same experience at different scales: Fast revises search within a task and records those revisions as adaptive traces of search, recovery, and reflection, while the slow contextual meta-learning harness uses recurring patterns in these traces to update the contextual scaffold that guides future exploration and recovery. In this way, experience from completed tasks informs adaptation on subsequent tasks.

4 Experiments

We evaluate ReTrace along three dimensions: recovery within a task, consolidation of recovery experience into a transferable cross-task prior, and the effect of that prior on later online adaptation. Together, these tests cover the two timescales in ReTrace and their interaction.

4.1 Experimental Setup

We study three questions: whether Fast recovers from unproductive within-task trajectories (RQ1), whether Slow consolidates prior experience into a reusable scaffold (RQ2), and whether the two processes reinforce each other across tasks (RQ3).

We use WebWalkerQA for online scaffold evolution and V-GEMS QA (Wang et al., 2026) for frozen transfer. On WebWalkerQA, we retain tasks passing the annotated-path reachability audit (Appendix A.3), allow up to four runs per task, and report $\mathrm{pass@1}$–$\mathrm{pass@4}$. Slow updates the scaffold every 10 completed tasks using only prior trajectories, recovery experiences, and evaluator feedback; each update applies to subsequent tasks. On V-GEMS, we freeze the final WebWalkerQA scaffold, run each task once, and report $\mathrm{pass@1}$. Evaluator feedback is available only after task execution.

We compare Slow with a structured Experience Bank under the same executor, Fast recovery mechanism, model, budget, and evaluator. The Bank retrieves evaluator-validated episodes and procedural skills at the task and subtask levels, whereas Slow consolidates accumulated experience into a compact, versioned policy. On V-GEMS, we additionally compare with MANGO, which exploits global website structure, and FocusAgent (Kerboua et al., 2026), which filters observations for task relevance. The Bank uses only experience from previously completed tasks, with its construction, retrieval, and transfer protocols detailed in Appendix A.4. All ReTrace configurations share the same base agent and vary only in whether Fast adaptation and the Slow-learned scaffold are enabled.

A task may contain multiple recovery episodes. We define $\mathrm{pass@}k$ as the fraction of tasks solved within k complete runs, with additional WebWalkerQA runs restricted to previously unsolved tasks.

	WebWalker	Fast	Evolved skill	Full ReTrace
Fast adaptation	×	✓	×	✓
Slow-learned skill	×	×	✓	✓

Table 1: ReTrace configurations. Full ReTrace combines task-internal Fast adaptation with the Slow-learned evolved skill.

We also stratify success by recovery count and report inference tokens, LLM calls, and tokens per successful task. All ReTrace variants use `qwen3.5-plus` with temperature 0.3, at most 30 steps per run, and `max_tokens=4096`; the evaluator uses the same model with temperature 0. Additional implementation details appear in the appendix. Full ReTrace performs best on both benchmarks, supporting the joint use of Fast adaptation and the learned scaffold. Sections that follow examine within-task recovery (RQ1), cross-task learning (RQ2), and their interaction (RQ3).

4.2 Overall Performance

Figure 3 compares the four ReTrace configurations on WebWalkerQA and frozen V-GEMS, along with external baselines on V-GEMS. On WebWalkerQA, Full ReTrace achieves 66.98% pass@4, exceeding WebWalker by 19.81 percentage points, Fast (65.09%), and Evolved skill (61.32%). On frozen V-GEMS, Full ReTrace achieves 55.00% pass@1, outperforming Frozen evolved skill (49.29%), Fast (47.14%), and WebWalker (45.71%), as well as the external baselines FocusAgent (51.43%) and MANGO (45.00%).

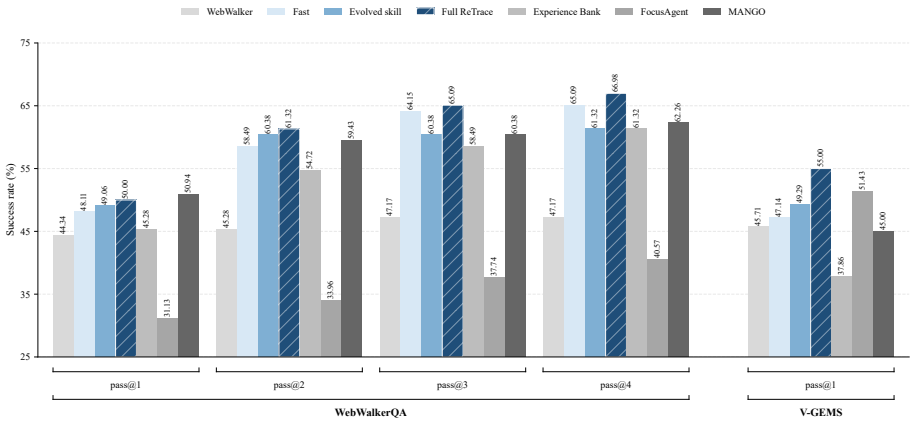


Figure 3: Overall success rate (%). Full ReTrace combines Fast adaptation with the evolved scaffold. On V-GEMS, the scaffold is frozen and results are reported as pass@1.

4.3 RQ1: Fast Recovers from Unproductive Search Within a Task

Fast adapts within a task by abandoning an unproductive continuation, recovering to an earlier state, and resuming search with a reflection-informed plan. Recovery appears frequently in successful trajectories. On WebWalkerQA, 25.49% of Fast's first-run successes involve at least one recovery, compared with 24.53% for Fast + evolved skill. On V-GEMS, recovery occurs in 12.12% of Fast's successes and 18.18% of Fast + evolved skill's successes. Table 2 reports first-run successful coverage from trajectories containing at most k recovery events.

On WebWalkerQA, most recovery-associated coverage is captured after a single recovery: Fast increases from 35.85% at $k = 0$ to 45.28% at $k = 1$, with smaller gains thereafter. On V-GEMS, the gains are distributed across more recovery events: Fast rises from 41.43% to 47.14%, while Fast + evolved skill rises from 45.00% to 55.00% between $k = 0$ and $k = 3$. These results show that successful search often depends on revising an unproductive continuation rather than continuing along

Dataset	Method	$k = 0$	$k = 1$	$k = 2$	$k = 3$
WebWalkerQA	Fast	35.85	45.28	46.23	48.11
	Fast + evolved skill	37.74	46.23	47.17	50.00
V-GEMS	Fast	41.43	44.29	45.00	47.14
	Fast + evolved skill	45.00	47.86	50.71	55.00

Table 2: First-run successful coverage (%) from trajectories with at most k recovery events, using all tasks as the denominator.

Method	WebWalkerQA				V-GEMS
	pass@1	pass@2	pass@3	pass@4	pass@1
WebWalker	44.34	45.28	47.17	47.17	45.71
Evolved skill	**49.06**	**60.38**	**60.38**	**61.32**	**49.29**

Table 3: Performance of Slow without Fast. On WebWalkerQA, the scaffold evolves from previously completed tasks; the final scaffold is frozen for V-GEMS.

the same search direction, with repeated recovery playing a larger role on V-GEMS. Appendix A.5 provides additional analyses of recovery-conditioned coverage and failure modes.

4.4 RQ2: Slow Learns a Reusable Search Prior Across Tasks

RQ2 isolates Slow by disabling Fast and evaluating whether experience from completed tasks can improve subsequent search through the learned scaffold. Table 3 shows that the scaffold consistently improves performance on WebWalkerQA. Evolved skill gains 4.72 percentage points at pass@1 and 14.15 points at pass@4 over WebWalker, with most of the cumulative gain already realized by pass@2. Thus, cross-task experience improves later search even without task-internal Fast adaptation. Slow also shifts successful search earlier in the trajectory. On WebWalkerQA, Evolved skill reaches 47.17% success within 10 recorded environment actions, already exceeding WebWalker's final first-run success. The learned prior therefore guides the executor toward productive trajectories with fewer interactions.

The scaffold itself evolves from task-specific navigation patterns toward reusable search principles, including exploiting information already present in the observation, switching branches when further exploration yields little information, validating interaction semantics before repeating actions, and stopping once sufficient evidence has been gathered to answer the question. These principles are task-agnostic rather than tied to a particular website or action sequence, allowing the final scaffold to transfer to V-GEMS even when it is frozen before evaluation. The improvement on V-GEMS therefore suggests that Slow learns a reusable search prior rather than merely memorizing task-specific navigation traces.

4.5 RQ3: Fast and Slow Reinforce Each Other Across Timescales

ReTrace couples Fast and Slow across timescales: Fast converts unproductive branches into structured recovery experience, while Slow consolidates recurring patterns into a persistent scaffold.

We first evaluate whether Fast-produced recovery experience improves Slow via a 2×2 factorial ablation (Table 4). Outer reflection yields gains only alongside structured recovery (+1.89 points at budget 30), whereas it impairs performance under raw-branch recovery (-1.89 points). Structured recovery outperforms raw-branch recovery by 7.55 points without outer reflection and by 11.32 points with it. This 3.77-point difference indicates a positive interaction: filtering raw execution noise allows Slow to consolidate clean recovery transitions into reusable guidance. Next, freezing the learned scaffold on target tasks demonstrates that Slow enhances Fast. The frozen scaffold boosts coverage over standalone Fast from 47.14% to 55.00% ($k = 3$), widening the gain from 3.57 to 7.86 points as recovery trajectories are incorporated. Within-run success also improves steadily to 55.00% (vs. 47.14% for Fast). Results confirm bidirectional coupling: Fast supplies structured signals for Slow to consolidate, while Slow provides priors that guide search and recovery in Fast.

Table 4: Factorial ablation of recovery representation and outer reflection on WebWalkerQA tasks (first-run success %).

	Structured recovery		Raw-branch recovery		Gain	
Budget	**Outer On**	**Outer Off**	**Outer On**	**Outer Off**	**Δ vs. best**	**Δ vs. worst**
10	**47.17**	43.40	34.91	37.74	+3.77	**+12.26**
20	**49.06**	48.11	38.68	40.57	+0.94	**+10.38**
30	**50.00**	48.11	38.68	40.57	+1.89	**+11.32**

4.6 Computational Characteristics and Failure Analysis

Figure 4 compares execution costs and budgeted success rates. Relative to WebWalker, Fast cuts environment actions by 4.5% at the cost of more LLM calls (+20.4%) and completion tokens (+26.1%), while Evolved skill reduces actions by 5.8% with 22.9% more tokens. Full ReTrace achieves the best efficiency balance: compared to WebWalker, it uses 10.4% more calls, 6.0% fewer tokens, and 12.4% fewer actions, effectively offsetting Fast's overhead.

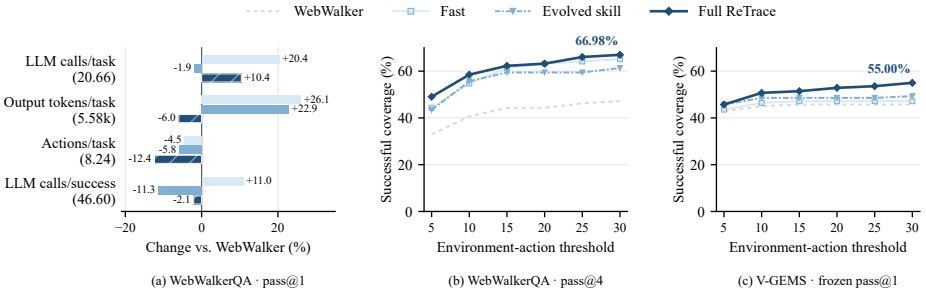


Figure 4: Execution cost and successful coverage. (a) Relative cost to WebWalker on WebWalkerQA pass@1 (absolute baseline costs in parentheses). (b,c) Successful task percentage within action budgets on WebWalkerQA pass@4 and frozen V-GEMS pass@1.

Across action budgets (5–30), Full ReTrace leads in average success rate on both WebWalkerQA (61.01% vs. Fast 58.81%, Evolved skill 56.45%, WebWalker 42.61%) and frozen V-GEMS transfer (51.55% vs. Evolved skill 48.21%, Fast 46.43%, WebWalker 45.12%), demonstrating sustained performance in both early search and extended trajectories. Residual failures mainly stem from missing golden pages (57.14% on WebWalkerQA; 76.19% on V-GEMS), with the remainder failing at final task completion despite finding evidence (Appendix A.5).

5 Conclusion

We introduced ReTrace, a two-timescale contextual meta-learning framework that connects task-local recovery with cross-task experience consolidation for web traversal. At the fast timescale, ReTrace detects unproductive search branches, retreats to an earlier decision point, restores branch-consistent context, and retains a compact reflection to guide rerouting. At the slow timescale, it consolidates evidence from completed recovery trajectories into a reusable contextual scaffold that shapes subsequent search and adaptation. On WebWalkerQA, Fast improves pass@4 from 47.17% to 65.09%, while combining Fast with the evolved scaffold reaches 66.98%. When the scaffold is frozen and transferred to V-GEMS, Full ReTrace achieves 55.00% pass@1, compared with 47.14% for Fast alone and 49.29% for the evolved scaffold alone. Together, these results suggest that recovery is useful not only for redirecting search within a task, but also for producing structured experience that can improve future behavior. ReTrace therefore provides a unified view of adaptation in which agents recover in order to learn and learn how to recover.

AI USE STATEMENT

In this work, we used generative AI tools solely for language editing and polishing, including improving grammar, clarity, and readability of the manuscript. We did not use generative AI tools for generating research ideas, conducting experiments, analyzing results, producing figures, or writing code, and other disclosure categories are not applicable to this work. We have reviewed all AI-assisted edits and verified that they did not alter the technical meaning, claims, or conclusions of the paper. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. In *International Conference on Learning Representations*, 2023.

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *International Conference on Learning Representations*, volume 2025, pp. 63707–63738, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30583–30598, 2022.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your llm secretly a world model of the internet? model-based planning for web agents. *Transactions on Machine Learning Research*, 2025.

Priyanshu Gupta, Shashank Kirtania, Ananya Singha, Sumit Gulwani, Arjun Radhakrishna, Gustavo Soares, and Sherry Shi. Metareflection: Learning instructions for language agents using past reflections. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8369–8385, 2024. doi: 10.18653/v1/2024.emnlp-main.477.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6864–6890, 2024. doi: 10.18653/v1/2024.acl-long.371.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Tianqing Fang, Zhenzhong Lan, and Dong Yu. OpenWebVoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27545–27564. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1336.

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. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1464.

Imene Kerboua, Sahar Omidi Shayegan, Xing Han Lù, Léo Boisvert, Megh Thakkar, Massimo Caccia, Jérémy Espinas, Alex Aussem, Véronique Eglin, and Alexandre Lacoste. Focusagent: Simple yet effective ways of trimming the large context of web agents. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=mINaJKSy7A.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024. doi: 10.18653/v1/2024.acl-long.50.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5295–5306, 2024. doi: 10.1145/3637528.3671620.

Haohao Luo, Jiayi Kuang, Wei Liu, Ying Shen, Jian Luan, and Yang Deng. Browsing like human: A multimodal web agent with experiential fast-and-slow thinking. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14232–14251. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.697.

Jinghao Luo, Yuchen Tian, Chuxue Cao, Ziyang Luo, Hongzhan Lin, Kaixin Li, Chuyi Kong, Ruichao Yang, and Jing Ma. From storage to experience: A survey on the evolution of LLM agent memory mechanisms. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 41622–41652. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.2069.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Haitao Mao, Guangliang Liu, Yao Ma, Rongrong Wang, Kristen Johnson, and Jiliang Tang. A survey to recent progress towards understanding in-context learning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 7317–7338. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.408.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

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

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Paloma Sodhi, S. R. K. Branavan, Yoav Artzi, and Ryan McDonald. Step: Stacked llm policies for web actions. *arXiv preprint arXiv:2310.03720*, 2023.

Weixi Tong, Yifeng Di, and Tianyi Zhang. Mango: Multi-agent web navigation via global-view optimization. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14209–14223. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.646.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 35151–35174. PMLR, 2023.

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

Xinjun Wang, Shengyao Wang, Aimin Zhou, and Hao Hao. See and remember: A multimodal agent for web traversal. *arXiv preprint arXiv:2603.02626*, 2026. URL https://arxiv.org/abs/2603.02626.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and Fei Huang. WebWalker: Benchmarking LLMs in web traversal. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10290–10305. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.acl-long.508.

Shiguang Wu, Yaqing Wang, and Quanming Yao. Why in-context learning models are good few-shot learners? In *International Conference on Learning Representations*, 2025b.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts LLM agents: An empirical study of experience-following behavior. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 623–645. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.27.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents. In *International Conference on Learning Representations*, volume 2025, pp. 97533–97565, 2025. URL https://openreview.net/forum?id=oWdzUpOlkX.

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.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4v(ision) is a generalist web agent, if grounded. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 61349–61385. PMLR, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946.

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 *International Conference on Learning Representations*, 2024.

A APPENDIX

A.1 RETRACE ALGORITHM

Algorithm 1 summarizes the two-timescale procedure in Section 3. The contextual scaffold remains fixed within each task window. Fast recovery jointly blocks the failed navigation edge, restores the navigation state, and revises the active context, while preserving each abandoned suffix in a separate record for subsequent slow learning.

Algorithm 1 ReTrace: Fast Recovery and Slow Contextual Meta-Learning

Require: Task windows $\{\mathcal{I}_k\}_{k=0}^{K-1}$; scaffold S_0; ledger $\mathcal{L}_0$; action, recovery, and scaffold budgets B_A, B_R, B_S; threshold δ
Ensure: Task answers $\{\hat{y}_i\}$, updated scaffold S_K, and updated ledger $\mathcal{L}_K$

for $k = 0, \ldots, K-1$ **do**
  $\mathcal{B}_k \leftarrow \emptyset$
  for $i \in \mathcal{I}_k$ **do**
    $(x, m, \tau, b_A, b_R) \leftarrow \text{Initialize}(q_i, B_A, B_R)$; $\mathcal{Z} \leftarrow \emptyset$; $\mathcal{R}_i \leftarrow [\,]$
    while task is unfinished and action budget remains **do**
      $(x, m, \tau, b_A, b_R) \leftarrow \text{SearchUntilRecovery}(q_i, x, m, \tau, b_A, b_R, \mathcal{Z}; S_k, \delta)$
      if task is unfinished and recovery is feasible **then**
        $(d, f) \leftarrow P(q_i, \mathcal{C}_i, \tau)$
        $g \leftarrow \text{OutgoingEdge}(\mathcal{C}_i, d)$; $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{g\}$
        append (τ, d, f) to $\mathcal{R}_i$ ▷ Separate from active context
        $x \leftarrow x_d$; $m \leftarrow \text{Update}(m_d, f)$; $\tau \leftarrow \tau_{\le d}$
        $b_R \leftarrow b_R - 1$; $b_A \leftarrow \text{RemainingActionBudget}()$ ▷ No action-budget refund
      end if
    end while
    $\hat{y}_i \leftarrow \text{SubmitAnswer}(q_i, m; S_k)$
    $\mathcal{H}_i \leftarrow \text{BuildHistory}(\mathcal{R}_i, \tau)$; $r_i \leftarrow \text{Judge}(q_i, \hat{y}_i, y_i)$
    $\mathcal{B}_k \leftarrow \mathcal{B}_k \cup \{(q_i, \mathcal{H}_i, \hat{y}_i, y_i, r_i)\}$
  end for
  $(\mathcal{P}_k, \mathcal{L}_{k+1}) \leftarrow \text{MetaReflect}(S_k, \mathcal{B}_{\le k}, \mathcal{L}_k)$
  $S_{k+1} \leftarrow \mathcal{U}(S_k, \mathcal{P}_k)$ subject to $\ell(S_{k+1}) \le B_S$
end for

The auxiliary operations in Algorithm 1 are implemented as follows. $\text{SearchUntilRecovery}$ executes search interactions, maintains candidate recovery points and their outgoing navigation edges, and updates state, context, trajectory, and budgets until the task terminates, the action budget is exhausted, or $V(q_i, x, m, b_A) < \delta$ with an available recovery point and sufficient recovery budget. The scaffold remains fixed throughout this process. Recovery blocks the selected outgoing edge for the remainder of the run, reuses the retained prefix without re-execution, and does not refund spent action budget. BuildHistory uses the archived episodes and final trajectory to construct the recovery traces in Equation 6 and the adaptation history in Equation 7, extracting each new suffix from the episode following its associated recovery. If no recovery occurs, the history contains only the final trajectory. Reference answers and grading feedback become available only after answer submission. The inner loop represents one complete run; evaluation retries repeat this run-level procedure and retain a separate adaptation history and outcome, as specified in Appendix A.4.

A.2 CASE STUDY: FAST ADAPTATION DURING WEB TRAVERSAL

This case study provides a trace-level view of fast adaptation on a WebWalkerQA task. The task asks for the number of instant-noodle packages donated by Uni-President after the December 21, 2023 Jishishan earthquake in Gansu. The answer is more than 32,000 packages. Figure 5 reproduces the five page-level trace panels from the accompanying execution. To keep the observation focused on the agent's interface, we show only the clickable buttons and omit the webpage screenshots.

Step 1: initial homepage decision.



(a) Step 1: Initial homepage decision (b) Step 2: Detecting a mismatched page (c) Step 3: Using the recovery reflection (d) Step 4: Locating the target article (e) Step 5: Answer submission

Figure 5: Trace panels for the case study. Each panel records the clickable-button observation, agent thought, action, and the change in active context; webpage screenshots are omitted from the textual trace below.

Observation

```
<button>Home<button>
<button>Statement on the New National Standard for Prepackaged Food
  Labels<button>
<button>Uni-President Receives 2024-2025 Food Safety
  Certificates<button>
<button>Love Relay: Uni-President and Disaster-Affected
  People<button>
[...]
```

Agent thought

```
The user is asking for specific donation details (number of instant
  noodle packs)
regarding the Gansu Jishishan earthquake on December 21, 2023.
  Looking at the
homepage buttons, there is a button labeled "Chinese text Chinese
  text
Chinese text" (Love Relay: Uni-President Enterprise and Disaster
  Area People Heart to
Heart) which seems highly relevant to disaster relief news. There is
  also a "Chinese text
Chinese text" (News Zone) and "Chinese text" (Corporate News). Given
  the specific title
mentioning "disaster area", clicking that button directly is the
  most efficient first
step to find the specific earthquake relief report.
```

Action and active-context update

```
Action: CONTINUE
visit_page(button='Love Relay: Uni-President and Disaster-Affected
  People')
activate_context: {} -> {step_1}
```

Step 2: detecting a mismatched page.

Observation

```
<button>Home<button> <button>Company Profile<button> <button>Main
  Markets<button>
<button>Management Team<button> <button>Brands<button> <button>News
  Zone<button>
[...]
```

Agent thought

```
Recovery reason: The current page discusses a 2025 earthquake relief
  effort in Tibet
(Dingri County), not the 2023 Gansu Jishishan earthquake requested.
  The specific
data point (number of noodle packs) is missing. While navigation
  buttons like 'News
Zone' exist to potentially find the correct article, the current
  trajectory has led to
an irrelevant topic, requiring a pivot rather than a simple
  continuation of the current
path. Reflection: The button labeled 'Love Relay: Uni-President and
  Disaster-Affected People' on the
homepage leads to a news article about a 2025 earthquake in Tibet,
  not the 2023
Gansu Jishishan earthquake. Do not assume buttons with generic
  disaster relief titles
correspond to the specific event in the query without verifying the
  date and location.
Instead, navigate to the 'News Zone' (News Zone) or use search
  functionality to locate
historical news from December 2023 regarding the Gansu earthquake.
```

Action and active-context update

```
Action: RECOVERY
recovery(state='step_1')
activate_context: {step_1} -> {step_1, step_2} -> {step_1,
  reflection_1}
```

Thus, the active context retains the homepage observation while adding a compact recovery reflection about verifying event date and location before continuing.

Step 3: using the recovery reflection.

Observation

```
<button>Uni-President Receives 2024-2025 Food Safety
  Certificates<button>
<button>Chinese text<button>
<button>Sunshine Home Program<button>
[...]
```

Agent thought

```
The previous attempt to click the "Chinese text" button led to
  irrelevant information
about a 2025 Tibet earthquake. The recovery reflection explicitly
  suggests navigating
to the "News Zone" (News Zone) to find historical news from December
  2023 regarding
the Gansu earthquake. I will follow this advice and click the "News
  Zone" button to
access the news archive where I can look for reports on the December
  21, 2023
Jishishan earthquake relief efforts.
```

Action and active-context update

```
Action: CONTINUE
visit_page(button='News Zone')
activate_context: {step_1, reflection_1} -> {step_1, reflection_1,
  step_3}
```

Step 4: locating the target article.

Observation

```
<button>Statement on the New National Standard for Prepackaged Food
  Labels<button>
<button>Uni-President Receives 2024-2025 Food Safety
  Certificates<button>
<button>All Hearts United: Uni-President Aids Gansu Earthquake
  Relief<button>
<button>Next Page<button>
[...]
```

Agent thought

```
I have successfully navigated to the "News Zone" (News Zone). The
  page displays a
list of news articles with dates. I can see an entry dated
  "2023/12/21" and a button
with the text "All Hearts United: Uni-President Aids Gansu
  Earthquake Relief" (All hearts united,
Uni-President aids Gansu earthquake relief front!). This title and
  date match the
query about the December 21, 2023, Gansu Jishishan earthquake. I
  will click this
button to read the full article and find the specific number of
  instant noodle packs
donated.
```

Action and active-context update

```
Action: CONTINUE
visit_page(button='All Hearts United: Uni-President Aids Gansu
  Earthquake Relief')
activate_context: {step_1, reflection_1, step_3} ->
                  {step_1, reflection_1, step_3, step_4}
```

Step 5: extracting the answer.

Observation

```
The opened article is titled ``All Hearts United: Uni-President Aids
  Gansu Earthquake Relief'' and is dated 2023/12/21. The article
  reports that Uni-President urgently dispatched more than 32,000
  instant-noodle packages to the earthquake-affected area in
  Jishishan County, Linxia Prefecture, Gansu.
```

Action and final answer

```
Action: FINAL_ANSWER

Final answer:
More than 32,000 instant-noodle packages.
```

The final page contains the evidence needed to answer the task. The article title and date identify the target event, while its text provides the exact donation amount. The agent extracts this evidence and submits the answer, completing the trajectory after the earlier recovery has redirected the search toward the date-indexed news archive.

A.3 Dataset Filtering: Motivation and Shared Audit Procedure

WebWalker Executor uses Crawl4AI to fetch web pages, converts HTML into Markdown, and maps extracted links to text buttons for navigation. This interface can encounter two types of environmental limitations. First, some interactive elements may be lost during fetching, content conversion, or button mapping, preventing the executor from performing actions available on the original page. Second, website defenses against automated access, including rate limits and human-verification challenges, may prevent the Crawl4AI-based WebWalker executor from accessing pages that remain available in an ordinary browser. Temporary network failures and changes to website content can also affect the executability of annotated navigation paths. Additional limitations arise from the executor's text-to-URL navigation abstraction: repeated button labels can produce ambiguous link mappings, relative links can be resolved differently from browser navigation, and extracted URLs do not reproduce stateful DOM-click semantics. These mismatches can remove browser-visible actions from the agent's action space or cause an annotated navigation step to execute differently from the corresponding browser interaction.

To reduce the influence of these environmental limitations on capability evaluation, we audit annotated-path reachability for WebWalkerQA and V-GEMS. The audit examines whether tasks have the conditions needed to obtain evidence through the current web interface, rather than whether a model can answer them correctly.

Both datasets follow the common auditing steps below.

1. **Read tasks and navigation annotations.** Extract each task's starting page, `root_url`, and annotated navigation paths, `golden_path`. Parse each path into an ordered sequence of button or link labels and apply WebWalker's button-text normalization rules.
2. **Check the starting page.** Fetch the starting page using WebWalker's `fetch_page` interface. Tasks with a missing starting URL, missing path annotations, or an unsuccessful starting-page fetch are marked as failing the audit.
3. **Replay annotated paths.** Extract links from the fetched HTML using the runtime `extract_links` interface and construct a mapping from button labels to target URLs. Look up each annotated button in sequence and fetch its target page, recording missing buttons, paths with no valid steps, and target-page fetch failures. Audit each annotated path separately when a task has multiple paths.
4. **Retry and record fetches.** By default, each fetch has a timeout of 60 seconds and is retried at most twice after failure. Fetch results for the same URL are cached and reused within an audit run.
5. **Produce filtering records.** Save task-level and path-level replay records, including failure steps, button labels, target URLs, fetch outcomes, and reasons, together with aggregate statistics. These records provide the basis for subsequent task retention and exclusion. Task retention is determined from these audit records under the dataset-specific filtering rules before agent evaluation.

The audit does not invoke an answering agent, select tasks based on model answer accuracy, search for alternative navigation routes, or perform additional DOM-level clicks.

The procedure below describes the common replay mechanism. Button-map updates and task-level decisions follow the respective filtering script; completing a replay is distinct from the final retention decision.

Fetch and replay operations are performed as follows. $\mathrm{PrepareTask}$ reads the start URL and annotated paths, initializes the task record, and fetches the root only when the annotations are valid; missing start URLs, empty path sets, and root-fetch failures are recorded explicitly. $\mathrm{FollowAnnotatedLink}$ records a missing-button failure when $b \notin M$; otherwise, it fetches $M[b]$ and returns the page together with a record of the button, URL, fetch outcome, and attempt metadata.

Algorithm 2 Annotated-Path Reachability Audit

Require: Tasks D; web interfaces; timeout $T = 60$ seconds; retry limit $R = 2$
Ensure: Audit records A and aggregate statistics S

$C \leftarrow \emptyset$; $A \leftarrow [\,]$ ▷ Shared fetch cache and audit records
for all $q \in D$ **do**
  $(P, \text{root}, a) \leftarrow \text{PrepareTask}(q; C, T, R)$
  if annotations are invalid or root fetch failed **then**
    append a to A; **continue**
  end if
  for all $p \in P$ **do**
    $B \leftarrow \text{NormalizePath}(p)$; page $\leftarrow$ root; $M \leftarrow \emptyset$; $\tau \leftarrow [\,]$
    for all $b \in B$ in order **do**
      $M \leftarrow \text{UpdateLinkMap}(M, \text{page})$
      $(\text{next}, z) \leftarrow \text{FollowAnnotatedLink}(b, M; C, T, R)$
      append step record z to τ
      if the button is missing or the fetch failed **then break**
      page $\leftarrow$ next
    end for
    record path p, trace τ, and completion status in a ▷ Empty paths are invalid
  end for
  aggregate path results under the dataset's filtering rules; append a to A
end for
$S \leftarrow \text{AggregateStatistics}(A)$; **return** (A, S)

All fetches share cache C. An uncached URL receives at most $R + 1$ attempts, each with timeout T, with waits between retries; success requires no exception and HTML and Markdown that are not both empty. Successful results and terminal failures are cached with their attempt metadata. UpdateLinkMap extracts and normalizes links according to the dataset's filtering script, with later entries overwriting earlier entries of the same normalized label. Each path has an independent trace and link map. Empty paths are invalid; nonempty paths are complete only when every annotated step succeeds. Task retention follows the dataset's filtering rules rather than path completion alone.

A.4 Evaluation and Aggregation Details

Experience Bank baseline. The Experience Bank serves as a retrieval-based alternative to our slow contextual meta-learning process. After each task, it stores evaluator-validated episodes together with procedural skills extracted at the task and subtask levels. For a new task, the baseline retrieves relevant entries and provides them as additional context to the executor. It uses the same executor, fast recovery mechanism, model, interaction budget, and evaluator as ReTrace to ensure a controlled comparison. Unlike Slow, which consolidates accumulated evidence into a compact and versioned policy, the Experience Bank retains and retrieves individual experiences without cross-experience consolidation.

A complete run starts from a task's initial state and produces one task-level outcome. A run with Fast enabled can contain several execution episodes separated by recovery events, each of which includes a task-local reflection. A retry is a new complete run; it is distinct from a recovery event within a run. On WebWalkerQA, each task allows up to four runs, while V-GEMS allows one run per task. A task is counted as successful when its recorded evaluator score equals one.

We use the LangChain `cot_qa` evaluator. In its prompt, the placeholders {`query`}, {`context`}, and {`result`} are populated with q_i, y_i, and $\hat{y}_i$, respectively. The exact evaluator prompt is reproduced below.

Evaluator Prompt

```
You are a teacher grading a quiz.
You are given a question, the context the question is about, and the
  student's answer. You are asked to score the student's answer as
  either CORRECT or INCORRECT, based on the context.
```

Evaluator Prompt *(continued)*

```
Write out in a step by step manner your reasoning to be sure that
  your conclusion is correct. Avoid simply stating the correct
  answer at the outset.

\medskip
Example Format:
QUESTION: question here
CONTEXT: context the question is about here
STUDENT ANSWER: student's answer here
EXPLANATION: step by step reasoning here
GRADE: CORRECT or INCORRECT here

\medskip
Grade the student answers based ONLY on their factual accuracy.
  Ignore differences in punctuation and phrasing between the student
  answer and true answer. It is OK if the student answer contains
  more information than the true answer, as long as it does not
  contain any conflicting statements. Begin!

\medskip
QUESTION: {query}
CONTEXT: {context}
STUDENT ANSWER: {result}
EXPLANATION:
```

The evaluator generates a free-text explanation followed by `GRADE: CORRECT` or `GRADE: INCORRECT`. The parser returns a dictionary with fields `reasoning`, `value`, and `score`. We map the parsed label `CORRECT` to $r_i = 1$ and `INCORRECT` to $r_i = 0$.

WebWalker uses neither Fast nor a persistent scaffold. Fast enables task-local recovery without a persistent scaffold. Evolved skill provides a learned scaffold without Fast, and Full ReTrace combines Fast with the evolved scaffold. The recorded Fast configurations allow up to three recoveries within a run. The agent uses `qwen3.5-plus`, temperature 0.3, a maximum of 30 steps per run, and `max_tokens=4096`. Evaluation uses the same model family with temperature zero. Slow updates the scaffold every ten completed WebWalkerQA tasks, applying each update to subsequent tasks. On V-GEMS, the evolved scaffold is frozen and task-local recovery does not update persistent state.

Let $y_{i,r} \in \{0, 1\}$ denote the recorded success of task i on complete run r, and let N be the number of evaluated tasks. The cumulative success indicator and reported rate are

$$s_i^{(k)} = \max_{1 \le r \le k} y_{i,r}, \qquad P_k = \frac{100}{N} \sum_{i=1}^{N} s_i^{(k)}. \tag{11}$$

Thus, pass@k measures observed cumulative task coverage over the ordered runs. The denominator remains N, including tasks that encounter runtime errors. WebWalkerQA tasks are processed in their original dataset order, without task reordering or difficulty-based scheduling. Together with frozen transfer on V-GEMS and the controlled Fast–Slow ablations, these evaluations examine performance across retry budgets, datasets, and method configurations.

For cumulative cost, a task contributes its first run and subsequent runs only while its earlier outcomes remain unsuccessful. If $r_i^\star$ is its first successful run, the retained runs through stage k are $1, \ldots, \min(k, r_i^\star)$; an unsolved task contributes all available runs through k. Later records for a task already solved are excluded from this accounting. The corresponding call, action, and token costs are reported in Figures 7 and 8.

A.5 Diagnostic Definitions and Residual Failures

We analyze successful trajectories, execution costs, and evidence coverage on 106 WebWalkerQA tasks and 140 V-GEMS tasks. WebWalkerQA results accumulate through $\mathrm{pass@}k$, $k \in \{1, 2, 3, 4\}$;

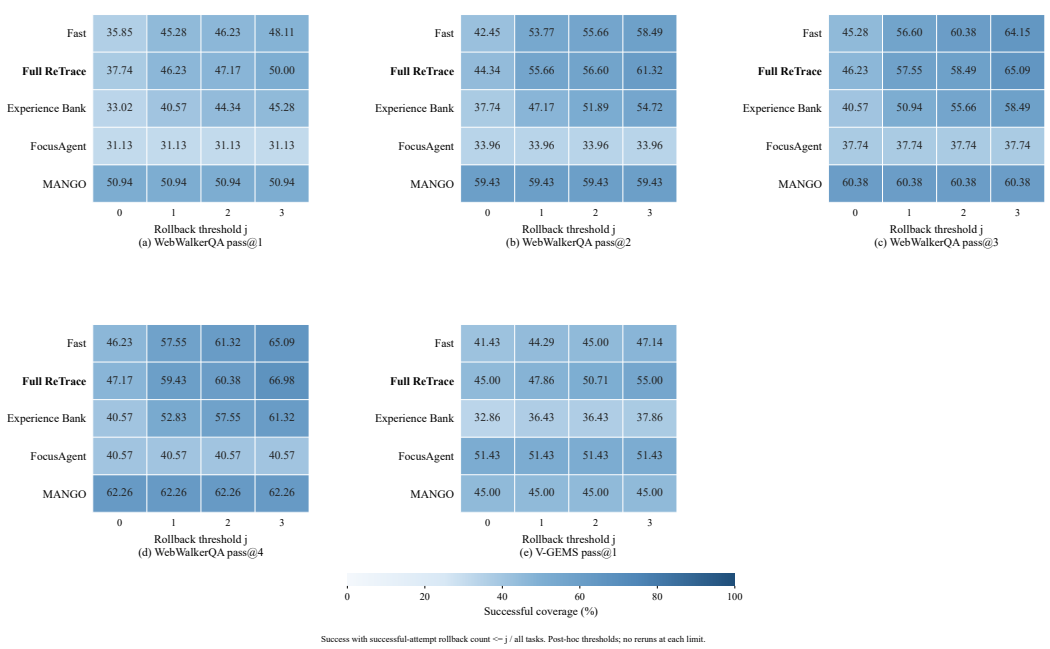


Figure 6: Successful coverage by recovery count. Each entry reports the percentage of all tasks solved by a recorded successful attempt containing at most j recovery events. Thresholds are applied post hoc to observed trajectories; runs are not repeated at each recovery limit. WebWalkerQA panels show cumulative outcomes through attempts 1–4; V-GEMS reports one attempt. NA entries denote unavailable measurements.

V-GEMS uses pass@1. At each stage, let $\mathcal{S}_k$ denote the successful tasks and $\mathcal{F}_k$ the residual failures, with $S_k = |\mathcal{S}_k|$ and $F_k = N - S_k$.

We further analyze successful coverage as a function of recovery count. Let $c_i^{(k)}$ denote the recovery count of the first recorded successful attempt for task i through stage k. Successful coverage at threshold j is

$$C_k(j) = \frac{100}{N} \sum_{i \in \mathcal{S}_k} \mathbf{1}\left[c_i^{(k)} \le j\right], \qquad j \in \{0, 1, 2, 3\}. \tag{12}$$

Figure 6 groups observed successes by these post-hoc thresholds. On WebWalkerQA, Full ReTrace's coverage increases from 37.74% at $j = 0$ to 50.00% at $j = 3$ under pass@1, and from 47.17% to 66.98% under pass@4. At the latter stage, 21 of its 71 solved tasks have successful attempts containing one to three recoveries. On V-GEMS, coverage increases from 45.00% to 55.00%, corresponding to 14 of 77 solved tasks. Successful trajectories involving recovery therefore account for a substantial share of task completion on both datasets.

Execution cost is further characterized across attempts. We aggregate recorded LLM calls, environment actions, and tokens over the retained attempts defined in Appendix A.4, including unsuccessful attempts. Per-task quantities divide these totals by N, while per-success quantities divide them by S_k. WebWalkerQA costs accumulate through the indicated stage; embedding operations are excluded from LLM usage.

Figure 7 shows that Full ReTrace uses fewer environment actions than WebWalker on WebWalkerQA: 7.22 versus 8.24 per task at pass@1, and 20.35 versus 34.50 at pass@4. At pass@4, it also uses fewer LLM calls per task, 61.27 versus 78.21, and fewer calls per success, 91.48 versus 165.80. The V-GEMS comparison exhibits a different allocation of computation: Full ReTrace uses 13.22 calls and 4.01 actions per task, compared with 8.21 calls and 1.56 actions for WebWalker, alongside a higher success rate of 55.00% versus 45.71%.

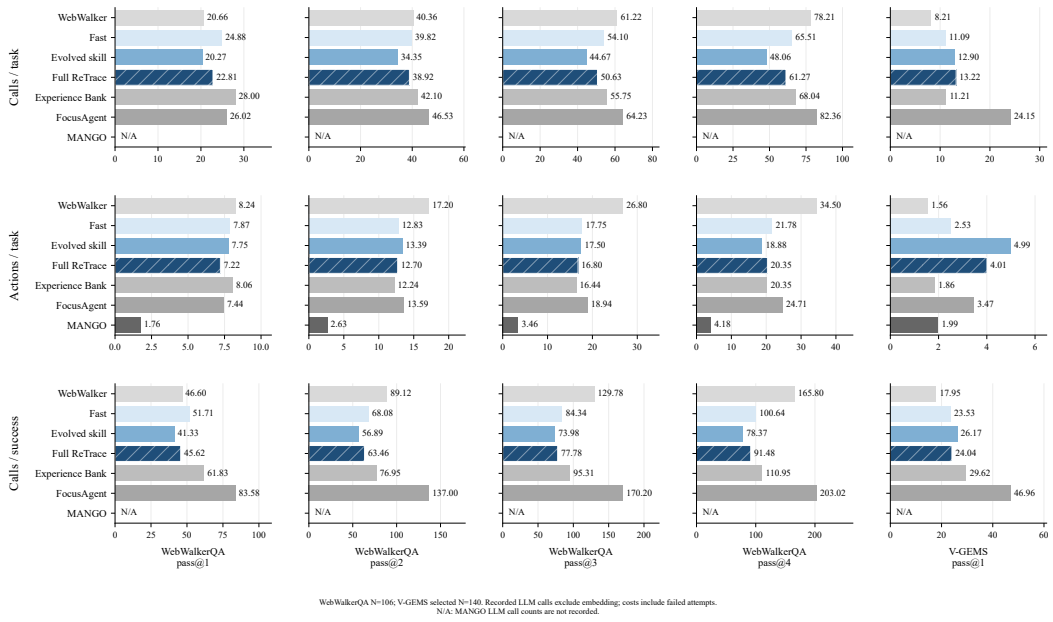


Figure 7: Recorded execution cost across evaluation stages. Rows show LLM calls per task, environment actions per task, and LLM calls per success. Totals include unsuccessful attempts and accumulate through each WebWalkerQA stage. Per-task quantities use all tasks as the denominator; per-success quantities use the cumulative number of solved tasks. LLM calls exclude embedding operations. Horizontal scales vary across panels. NA entries denote unavailable measurements.

Figure 8 separates input-token consumption from generated output. On WebWalkerQA pass@4, Full ReTrace uses 14.65k output tokens per task compared with 18.43k for WebWalker, while input usage is 504.73k versus 268.92k. Its total token cost per success is consequently higher, at 775.40k versus 609.19k. Relative to Fast, Full ReTrace uses fewer input tokens, output tokens, and total tokens per success at every reported WebWalkerQA stage. These comparisons reveal distinct aspects of execution efficiency: Full ReTrace reduces calls and actions relative to WebWalker at pass@4, while processing more input tokens.

Evidence coverage and residual failures are analyzed as follows. For task i, let G_i denote its annotated golden pages and $H_i^{(k)} \subseteq G_i$ the golden-page URLs recorded as hit through stage k. Full coverage means that all annotated pages are hit; partial coverage means that some but not all are hit. Any-hit coverage over all tasks is

$$Q_k = \frac{100}{N} \sum_{i=1}^{N} \mathbf{1}\left[|H_i^{(k)}| > 0\right]. \tag{13}$$

For any task set $\mathcal{D}$, macro coverage averages the fraction of golden pages hit:

$$M_k(\mathcal{D}) = \frac{100}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \frac{|H_i^{(k)}|}{|G_i|}. \tag{14}$$

Residual-failure statistics use $\mathcal{D} = \mathcal{F}_k$, so their denominator is the number of tasks still unsolved at that stage.

The evidence-coverage metric counts successful tasks together with residual failures having at least one golden-page hit:

$$E_k = \frac{100}{N} \left(S_k + \sum_{i \in \mathcal{F}_k} \mathbf{1}\left[|H_i^{(k)}| > 0\right] \right). \tag{15}$$

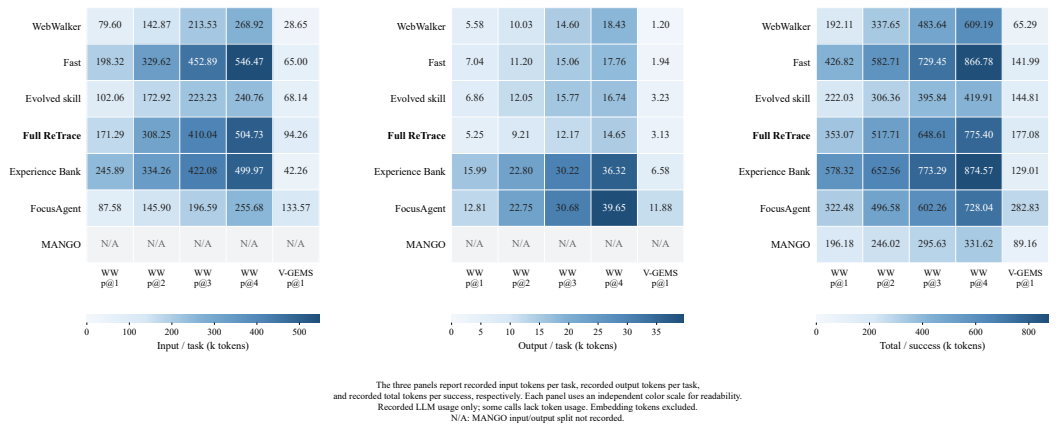


Figure 8: Recorded LLM token usage, in thousands of tokens. Panels report input tokens per task, output tokens per task, and total tokens per success. Usage includes unsuccessful attempts and accumulates through the indicated WebWalkerQA stage. Each panel uses an independent color scale. NA entries denote unavailable measurements.

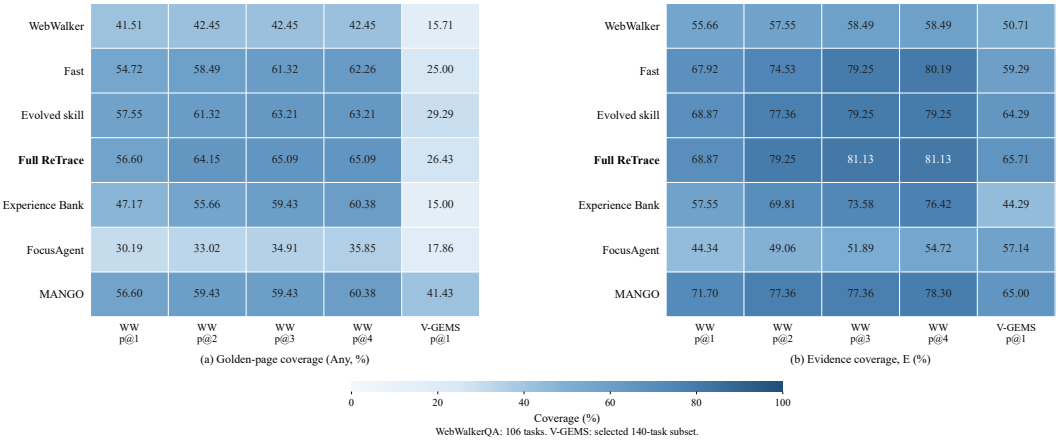


Figure 9: Golden-page and evidence coverage. Left: percentage of all tasks with at least one recorded golden-page URL hit, Q_k. Right: percentage of tasks that either succeed or remain unsuccessful with at least one golden-page hit, E_k. Both panels use all tasks as the denominator and report raw percentages. WebWalkerQA outcomes and hits accumulate through the indicated evaluation stage. NA entries denote unavailable measurements.

Figure 9 presents golden-page any-hit coverage Q_k and evidence coverage E_k separately. On WebWalkerQA pass@4, Full ReTrace achieves 65.09% any-hit coverage and 81.13% evidence coverage, compared with 42.45% and 58.49% for WebWalker.

Among Full ReTrace's 35 residual WebWalkerQA failures, 12 have full golden-page coverage, three have partial coverage, and 20 have no recorded hit. On V-GEMS, the corresponding counts are six, nine, and 48 among 63 residual failures. The remaining errors therefore span both tasks without recorded evidence hits and tasks that remain unresolved despite hitting annotated pages. The former group is larger on both datasets, especially V-GEMS.

On V-GEMS, MANGO has higher any-hit coverage than Full ReTrace, at 41.43% versus 26.43%, but lower task success, at 45.00% versus 55.00%. Their evidence-coverage scores are similar, at 65.00% and 65.71%, reflecting different balances between successful completion and evidence hits among residual failures.

Additional diagnostic definitions are introduced to facilitate further analysis. Answer production counts tasks with at least one recorded nonempty answer through stage k, regardless of correctness. Let $a_i^{(k)}$ be one if task i has such an answer in any recorded run through stage k, and zero otherwise. Then

$$A_k = \frac{100}{N} \sum_{i=1}^{N} a_i^{(k)}. \tag{16}$$

For a fixed attempt aggregation, let $U_k(b)$ denote the percentage of tasks with at least one successful run through stage k using at most b recorded environment actions, and let $P_k = 100S_k/N$ denote final success rate. The action threshold is applied to each run separately. Budgeted success averages the six reported environment-action thresholds:

$$B_k = \frac{1}{6} \sum_{b \in \{5,10,15,20,25,30\}} U_k(b). \tag{17}$$

Long-horizon gain is $P_k - U_k(10)$, measured in percentage points, and summarizes additional successful coverage beyond the ten-environment-action threshold.

A.6 Runtime Prompts and Context Construction

This appendix records the prompt components associated with the execution and recovery functions in Section 3. English source text is reproduced in monospaced text with presentation-only line wrapping. Markdown markers, where present, are shown literally rather than rendered. Chinese source text is identified as an English translation. Templates replace task-dependent values with angle-bracketed placeholders; these placeholders are not literal observations used in the experiments. The accompanying `prompt_skill_sources/` directory preserves original-language text and a source manifest.

A.6.1 Executor protocol and task header

The executor selects ordinary navigation actions and produces the final answer. Its fixed system protocol is:

Executor System Prompt

```
You are a WebWalker executor. Follow this protocol exactly.

Use this format:

Question: the input question you must answer
Thought: reason about the next ordinary navigation action
Action: the action to take, chosen only from the tools in the task
  header
Action Input: valid JSON for that action
Observation: the result of the action
... (repeat this Thought/Action/Action Input/Observation cycle only
  within the action budget shown in the current task header)

Hard constraints:
- Take an ordinary navigation action until sufficient information
  has been gathered, then answer exactly as: Final Answer: <your
  answer>.
- Branch evaluation, recovery and state selection are external
  controller responsibilities. Do not output, request, simulate, or
  mention them.
- If the task header contains recovery experience or previous
  attempts, avoid repeating those failed choices and take a
  materially different ordinary navigation action.
- Do not refuse the task; continue exploration within the available
  action budget.
- Action Input must be valid JSON.
```

When a task-solving skill is enabled and nonempty, the recorded system message places its text in a `Task Solving Skill` module before the `Executor Fixed Scaffold` module. Without a task skill, the fixed protocol remains. The initial user message supplies the question, root page, available tools, and action budget. Its task-dependent fields can be represented as follows:

Executor Task Header Template

```
Task Header
Question and initial context:
query: <question>
official website: <root URL>
Observation: <initial page observation>

Root URL: <root URL>
Available tools: <tool descriptions and argument schemas>
Begin!
Action budget: total=<total>, used=<used>, remaining=<remaining>
```

The recorded navigation interface is `visit_page`, with a JSON argument containing `button`. The button text must match an item in the current observation. The template above summarizes the field layout; the system protocol above preserves the runtime wording.

A.6.2 Continuation Assessment

The branch evaluator implements the continuation assessment V in Equation 1. It returns a score rather than choosing an action. The recorded controller threshold is $\delta = 5$, with recovery triggered when the score is strictly below this value. The following is an English translation of the Chinese system prompt:

Continuation Judge Prompt

```
You are WebWalker's branch evaluator. Judge only whether the current
  trajectory is worth one more ordinary navigation action, given the
  supplied evidence and remaining resources.

Do not select links, tools, recover, states, or the next step. Do
  not discuss thresholds.

continuation_score is an integer from 0 to 10: 0 means almost no
  reason to continue; 10 means a strong and concrete reason to
  continue.

Return only the following JSON:
{
  "continuation_score": 0,
  "continuation_evidence": "",
  "risk_signals": "No material risk signals"
}

For scores of at least 5, continuation_evidence must be nonempty:
  provide concrete facts from the page or trajectory, or cite the
  budget and explain a resource-based reason why there is no clear
  dead end. risk_signals records actual risks or "No material risk
  signals". Return JSON without any additional content.
```

The user-message template contains the following fields:

Continuation Judge Input Template

```
Target Task: <question>
Current Observation: <current page observation>
```

Continuation Judge Input Template *(continued)*

```
Recent Active Trace: <serialized recent active trajectory>
Confirmed Evidence: <confirmed evidence, or No confirmed evidence
  yet.>
No-progress visit count: <count>
Action Budget: total=<total> / used=<used> / remaining=<remaining>
Recover Budget: total=<total> / used=<used> / remaining=<remaining>
```

A.6.3 Recovery-point selection and failure reflection

The recorded planner jointly supplies the recovery point and the task-local reflection in Equation 2. Its output fields are `back_idx` and `experience`, respectively. The fixed system prompt is:

Recovery-Point Selection and Reflection Prompt

```
You select where a WebWalker executor should roll back after a
  critic determined that the current state is unpromising.

Select exactly one state from the supplied state list. You may not
  invent a URL or state.
Selecting a state permanently prunes its supplied outgoing failed
  visit_page edge (source URL -> target URL) from the action space
  after restoration. Prefer the deepest state whose supplied
  outgoing failed edge is itself justified to prune; choose an
  earlier state only when the broader branch, including that earlier
  edge, is wrong.
Summarize the failed suffix as a self-contained lesson that helps
  the executor choose a materially different action after
  restoration.

Return one JSON object with exactly these fields:
{
  "analysis": "brief reason for selecting the state",
  "back_idx": 0,
  "experience": "self-contained lesson from the failed branch"
}
```

The planner receives the saved states and the failed trajectory information. Its output is consumed by the controller; ordinary navigation resumes under the executor protocol. The reflection text is therefore a task-local output of this prompt, rather than a manually filled example in the system message.

A.7 Recorded Skill Update Protocol

The recorded update harness provides concrete components for experience analysis and contextual scaffold update in Section 3. It separates research, synthesis, and review. The excerpts below reproduce selected clauses from the recorded system prompts. Full recorded system-prompt variants are preserved in the accompanying source directory.

A.7.1 Harness construction and orchestration

Skill evolution is implemented as a stateful, windowed update harness around the executor. The harness begins from an empty published skill, `v000`, and processes tasks sequentially. After every fixed window of completed and evaluated tasks (ten tasks in our WebWalkerQA runs), execution reaches an evolution barrier: the next task is not started until the pending update window reaches a terminal decision. Consequently, a task can only observe the skill version published by preceding windows.

Each evolution window operates over an immutable snapshot containing the completed-task records accumulated so far, their evaluations, the currently published skill, selected executor-side runtime

code, and persistent analysis state. The update is decomposed into three role-isolated stages: *research*, *synthesis*, and *review*, whose recorded prompts and stage-specific constraints are detailed below. Across windows, these stages consolidate recurring traversal, recovery, and reflection guidance into the published scaffold.

For windows containing any high-confidence change, synthesis is prevented from reading the parent skill and reconstructs the candidate from the evidence-backed proposal and document skeleton; when all proposed changes are low-confidence wording refinements, it may instead read the parent and make only the permitted local edits.

A `REVISE` decision routes explicit issues back to either research or synthesis in a fresh conversation. The implementation permits at most two such revisions after the initial attempt. Artifacts from already accepted stages are carried forward when possible, whereas the targeted stage is rerun. Only `CANDIDATE_WINS` can publish a new skill. Before publication, deterministic checks verify the parent and candidate identities, evidence references, structural constraints, and token budget. `PARENT_WINS` closes the window without changing the active skill, while `REJECT` terminates the window without publication.

Published skills are immutable versioned artifacts. Publication is committed before the active version is advanced, and the newly active skill is injected into the executor only for subsequent tasks. The analysis ledger persists across evolution windows but is not included in the executor context. The evolving contextual scaffold is serialized as the task-solving skill injected into the executor. The continuation assessment and recovery planner retain their fixed prompts across evolution windows. The harness also records per-window inputs, research and synthesis artifacts, review decisions, model/tool traces, and publication metadata, allowing an interrupted run to resume from durable artifacts while preserving the same dataset ordering and previously published versions.

The publication budget is defined to constrain the amount of information released. The skill budget is frozen before online evolution using the `cl100k_base` tokenizer. For the reported WebWalkerQA evolution run, the human-written reference contains 1,239 tokens, from which the harness records a target budget of 1,115 tokens. The enforced publication ceiling in every evolution window is this fixed 1,115-token target; the 1,239-token reference length is retained as frozen budget metadata rather than used as the per-window publication limit. The ceiling therefore neither expands with a previously published skill nor contracts when an intermediate version is short. Candidates must be submitted as complete documents and are never mechanically truncated.

A.7.2 Research: evidence-grounded changes

The research stage reads completed-task materials and the current skill, maintains an analysis ledger, and proposes changes linked to observed records. The following excerpts specify its evidence and abstraction requirements:

Skill Research Prompt (Excerpts)

```
You are the research member of an online web-task skill evolution
  team.

Investigate only completed tasks. Read window metadata, completed
  records, the current runtime skill, approved runtime code, and
  durable analysis. Audit original observations before judge
  reasoning or gold. Distinguish environment, access, parser, tool,
  evaluation, and genuine decision-policy failures. Do not replay
  executor tasks or components.

Maintain a detailed analysis ledger outside the runtime skill:
  supported principles, failure clusters, semantic duplicates,
  superseded principles, evaluation disputes, and unresolved
  hypotheses. A historical failure under an older skill is not a new
  gap if the current principle already covers it.
```

Skill Research Prompt (Excerpts) *(continued)*

```
Abstract every proposed principle to the domain-independent decision
  policy, not the surface case.

If two successful tasks share a policy at the abstract level, record
  ONE merged principle, not two domain-specific copies. Derive the
  wording from the observed trajectories themselves; do not import
  phrasing from any external reference skill.
```

Each change record includes an operation, historical skill version, current coverage, intended behavioral change, reason, confidence, and evidence references. Evidence references identify completed tasks and JSON pointers into their records. The allowed operations are `add`, `modify`, `delete`, `merge`, `reorder`, and `restore`. Concrete task identifiers remain in the ledger rather than in the runtime skill.

A.7.3 Synthesis: candidate construction

The synthesis stage produces a complete candidate skill from the research changes. Recorded prompts distinguish general synthesis from a low-confidence refinement mode. The general mode reconstructs the skill from the ledger and evidence without reading the parent text; its prompt begins:

Skill Synthesis Prompt (Excerpt)

```
You are the synthesis member of an online web-task skill evolution
  team.

Build a COMPLETE runtime skill from scratch from the research
  ledger, evidence-backed changes, skeleton, and numerical budget.
  You cannot read the parent skill: this prevents incremental
  wording anchors. Preserve only portable principles that change
  executor decisions. Put case details, answers, identities, URLs,
  audit rationale, and unresolved analysis outside the runtime skill.

Every retained rule needs an observable trigger, a concrete action,
  and a useful stopping condition. Merge semantic duplicates. Do not
  pad. Do not mechanically truncate. The candidate must fit the
  supplied hard publication budget, a fixed ceiling supplied each
  window. Use measure_candidate_text before submission if useful.
```

In low-confidence refinement mode, the synthesis stage instead reads the parent and restricts changes to small edits. The corresponding prompt states:

Low-Confidence Refinement Prompt (Excerpt)

```
You are the synthesis member of an online web-task skill evolution
  team, operating in LOW-CONFIDENCE REFINEMENT mode.

Every change proposed this window is low-confidence and therefore
  wording-level only, so you build the candidate by lightly editing
  the parent, not by rebuilding from scratch. First read
  current/task_solving.md --- the parent skill --- and reproduce it
  VERBATIM except for the specific wording-level micro-adjustments
  named in the research change records. Preserve the parent's title,
  section headings, principle set, ordering, language, and every
  line not touched by a change exactly as written. Do not rewrite
  prose into bullets or bullets into prose, do not translate or
  switch languages, do not introduce a new title, and do not add,
  delete, merge, or restructure any principle. Apply only the
  modify/reorder edits the ledger specifies: sharpen a phrasing,
  tighten a trigger, add or clarify a stopping clause, or reorder
```

Low-Confidence Refinement Prompt (Excerpt) *(continued)*

```
  existing principles. The result must read as the parent with a few
  small edits, not as a fresh document.
```

The candidate submission includes its complete Markdown text, an analysis, and change records carrying the original evidence references. The low-confidence mode permits `modify` and `reorder`; the full recorded mode variants are kept separately in the source bundle.

A.7.4 Review and publication

The review stage compares the candidate with the immutable parent and checks the supporting records. Its bootstrap criterion provides a concise statement of the required structure of a learned principle:

Skill Review Prompt (Excerpt)

```
Bootstrap case: when the parent is empty, do not reject solely for
  lacking a decision advantage over the parent --- an empty parent
  gives no guidance, so any sound principle is already an advantage.
  For an empty parent, choose CANDIDATE_WINS when every candidate
  principle has an observable trigger, a concrete action, a stopping
  condition, and evidence grounded in a completed record; still
  choose PARENT_WINS or REVISE for principles that are task-specific
  facts, plausible prose with no decision effect, or unsupported by
  evidence.
```

Approved candidates are published as the next scaffold version and become available to subsequent tasks.

A.8 Evolved Contextual Scaffold

Slow progressively consolidates experience from completed tasks into an evolving contextual scaffold. The scaffold is serialized as a task-solving skill and organizes reusable guidance into three functional skill classes introduced in the main text: Traversal Skill, Recovery Skill, and Reflection Skill. We show three representative published versions that mark this evolution: `v001`, the first nonempty scaffold; `v006`, which introduces explicit recovery guidance; and `v010`, the final scaffold used for frozen transfer.

A.8.1 Early published skill

The Full ReTrace evolution record starts from an empty `v000`. The first approved version, `v001`, contains 198 tokens under the recorded tokenizer. It establishes an initial traversal policy together with a verification check. Its body is reproduced below; only the four Chinese section headings are translated into English, while the original Markdown structure is retained.

Early Skill (v001)

```
## Overall objective

Provide a domain-independent methodology for navigating web
  structures to locate specific information based on question type
  and page organization patterns.

## Responsibilities

When questions ask for time, location, or topic details of scheduled
  items (lectures, talks, sessions, presentations), navigate to
  schedule or program sections. Stop when the page displays a
  structured timeline or event list containing the requested entity.
```

Early Skill (v001) *(continued)*

```
## Procedure

1. Identify if the question requests temporal, spatial, or topical
  information about a scheduled event item.
2. Locate and navigate to the website's schedule, program, or agenda
  section.
3. Scan for structured timelines or session lists.
4. Extract the requested information once the target entity appears
  in the listing.
5. Cease navigation when the entity is found with complete details.

## Quick review

Verify the located page contains a structured event listing with the
  requested entity's time, location, or topic clearly displayed
  before concluding the task.
```

This first version captures the initial form of Traversal Skill by mapping the requested information to an appropriate website section and specifying when navigation should stop. Its quick-review clause provides an early Reflection Skill by verifying that the retrieved content satisfies the task before submission.

A.8.2 Intermediate skill with recovery guidance

By `v006`, the scaffold expands from initial traversal to recovery from unproductive search directions. It adds complementary-section search, alternative-section exploration, and stronger verification and stopping rules.

Intermediate Skill (v006)

```
# Web Information Retrieval Skill

## Overall objective

Efficiently locate and extract specific requested details from
  structured web content by navigating to relevant sections,
  cross-referencing when information spans categories, and
  recovering from incorrect initial navigation.

## Responsibilities

- Identify the type of information requested (entities, policies,
  schedules, attributes, or supporting documentation)
- Navigate to website sections likely containing that information
  type
- Cross-reference multiple related sections when the query involves
  combined categories
- Try alternative relevant sections if initial navigation does not
  contain the specific requested detail
- Extract only the specific detail requested, not just related
  information
- Stop when the page displays structured content containing the
  exact requested entity or detail

## Procedure

1. **Analyze the request**: Determine what type of information is
  being sought and whether it may span multiple categories (e.g.,
```

Intermediate Skill (v006) *(continued)*

```
  sponsorship combined with visa support requires checking both
  funding and policy sections).

2. **Initial navigation**: Visit the website section most directly
  matching the primary information type.

3. **Evaluate content**: Check if the current page contains the
  SPECIFIC requested detail. If only related but incomplete
  information is present:
    - Identify complementary sections that might contain the missing
  aspect
    - Visit those related sections to cross-reference
    - If the initial section was incorrect, try alternative relevant
  sections

4. **Extract and verify**: Once the specific detail is located in
  structured content, extract it precisely.

5. **Stopping condition**: Stop only when the page displays
  structured content containing the exact requested entity or detail
  that can be extracted. Do not stop for pages with only
  tangentially related information.

## Quick review

- Did I check multiple sections when the query involved combined
  categories?
- Did I try alternative sections when the initial navigation failed
  to yield the specific detail?
- Is the stopping condition met by finding the exact requested
  detail, not just related content?
```

The added cross-section and alternative-section strategies instantiate Recovery Skill by guiding how search should change after the current direction fails to provide the required evidence. The expanded quick review strengthens Reflection Skill through checks on evidence completeness, search alternatives, and stopping.

A.8.3 Final published skill and frozen deployment

The final published version, `v010`, contains 554 tokens and integrates the three functional skill classes into the scaffold used for frozen transfer. It extends traversal with homepage navigation cues, recovery with alternative-path selection, and reflection with context verification and stronger stopping conditions.

Final Skill (v010)

```
# Web Information Retrieval Skill

## Overall objective
Efficiently locate and extract specific requested details from
  structured web content by navigating to relevant sections,
  cross-referencing when information spans categories, and
  recovering from incorrect initial navigation.

## Responsibilities
- Identify the type of information requested (entities, policies,
  schedules, attributes, or supporting documentation)
- Navigate to website sections likely containing that information
  type
```

Final Skill (v010) *(continued)*

```
- Cross-reference multiple related sections when the query involves
  combined categories
- Try alternative relevant sections if initial navigation does not
  contain the specific requested detail
- Extract only the specific detail requested, not just related
  information
- Stop when the page displays structured content containing the
  exact requested entity or detail, after verifying the content
  matches the query's specified context (product/version/year)

## Procedure
1. **Analyze the request**: Determine what type of information is
  being sought and whether it may span multiple categories (e.g.,
  sponsorship combined with visa support requires checking both
  funding and policy sections).

2. **Initial navigation**: Visit the website section most directly
  matching the primary information type, using homepage
  announcements or news sections as hints about information location
  when available. When homepage news entries explicitly mention
  which section contains target information, navigate directly to
  that named section rather than inferring from information type
  alone.

3. **Evaluate content**: Check if the current page contains the
  SPECIFIC requested detail. When evaluating a page that presents a
  list or archive of items, check if any listed entry titles or
  summaries contain keywords matching the query. If no matching
  entries appear on the current page, prioritize exploring
  alternative navigation paths over paginating through multiple
  pages of the same list. If only related but incomplete information
  is present:
    - Identify complementary sections that might contain the missing
  aspect
    - Visit those related sections to cross-reference
    - If the initial section was incorrect, try alternative relevant
  sections

4. **Extract and verify**: Once the specific detail is located in
  structured content, verify that the page context
  (product/version/year) matches the query's specified context, then
  extract it precisely.

5. **Stopping condition**: Stop immediately when the page displays
  structured content containing the exact requested entity or detail
  that can be extracted and all requested aspects are satisfied. Do
  not stop for pages with only tangentially related information, and
  do not continue searching for unrequested related information once
  all requested details are found.

## Quick review
- Did I check multiple sections when the query involved combined
  categories?
- Did I try alternative sections when the initial navigation failed
  to yield the specific detail?
- Is the stopping condition met by finding the exact requested
  detail with matching context, not just related content?
- Did I stop immediately after finding all requested aspects rather
  than continuing to search?
```

In the final scaffold, Traversal Skill guides section selection, evidence acquisition, and the use of explicit homepage cues. Recovery Skill guides cross-section search, alternative-path exploration,

and departure from unproductive list traversal. Reflection Skill guides executor-side verification, context matching, and stopping. During Fast adaptation, the fixed recovery planner produces the task-local failure reflection, while the learned scaffold guides subsequent executor behavior.

The `inner_outer_frozen` executor snapshot on V-GEMS contains this same `v010` skill body before the fixed executor protocol. Its configured skill-file hash matches the published `v010` metadata, and task-skill evolution is disabled for that run. The frozen-transfer configuration therefore uses the final scaffold evolved on WebWalkerQA.

The evolution example illustrates how the process unfolds over time. The evolution progresses from an initial traversal and verification policy in `v001`, to explicit recovery guidance in `v006`, and finally to the integrated traversal, recovery, and reflection guidance in `v010`. Across these revisions, Slow converts recurring experience from completed tasks into reusable guidance that shapes subsequent search and adaptation.