NO RESET: EVALUATING AND TRAINING PERSISTENT AGENTS ON TASK SEQUENCES

Anonymous authors
Paper under double-blind review

ABSTRACT

AI agents are typically evaluated one task at a time, yet deployed agents operate in persistent sessions where conversation history, memory, files, and prior experience carry across tasks. We study this mismatch by introducing sequential variants of GAIA2 and SWE-Bench Pro, in which agents solve sequences of up to 40 tasks within a single persistent session. Our experiments show that persistence has a model-dependent effect on task success. Opus 4.6 and GPT-5.4 remain near their independent baselines, whereas DeepSeek v4 Pro loses 5.9 percentage points on average. More consistently, persistence changes how every model behaves: agents use up to 60% fewer output tokens, repeat less exploratory work, and, on GAIA2, increase Python's share of tool calls by up to $4\times$. Trace analyses reveal both sides of this adaptation: agents reuse knowledge and customized tooling across tasks, but they can also under-explore new tasks and carry unchecked assumptions forward. To improve model capability in this setting, we introduce Sequential RL (Seq-RL), which trains on persistent multi-task rollouts with fine-grained credit assignment. Trained only on GAIA2-style data disjoint from the evaluation set, Seq-RL substantially narrows the sequential gaps on both GAIA2 splits. It also transfers zero-shot to SWE-Bench Pro, where it outperforms a single-task RL baseline trained on the same data, despite receiving no software-engineering data during RL. In short, **sequential evaluation exposes new dimensions of agent reliability and in-context learning, while Seq-RL provides a promising training method for more robust long-lived agents.**

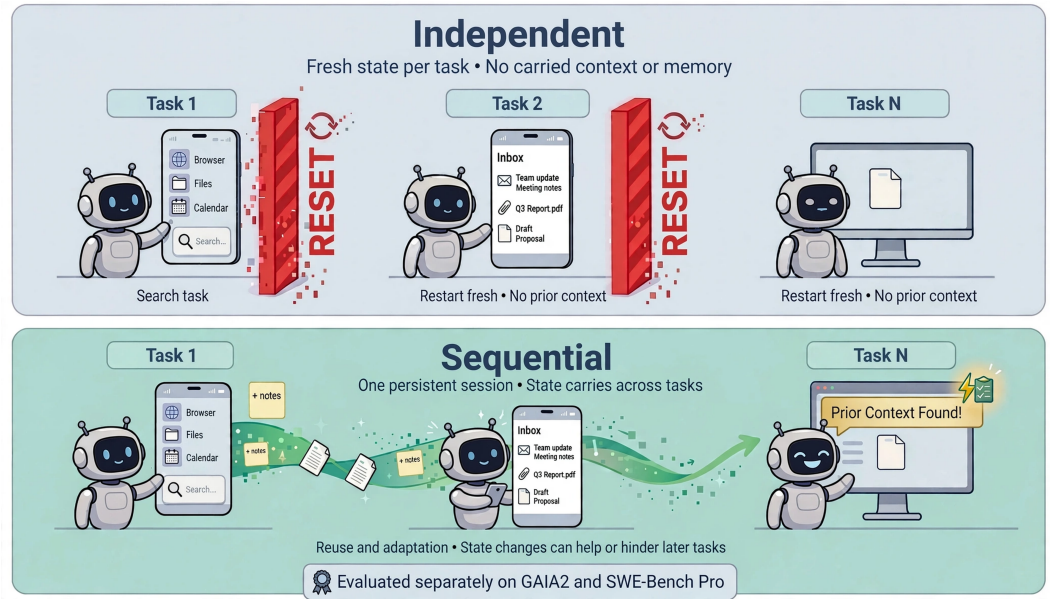


Figure 1: **Independent versus sequential agent evaluation.** The same tasks run in fresh or persistent sessions. Conversation and files carry over; GAIA2 also preserves memory and environment state, whereas SWE-Bench Pro resets its repository per task.

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.

1 INTRODUCTION

Today's agents operate in persistent sessions, from coding assistants (Claude Code, Codex) to personal agents (OpenClaw Team, 2026; Nous Research, 2026), accumulating conversation history, files, memory, and experience across tasks. Yet persistence is fragile: performance degrades with longer contexts (Liu et al., 2023; Du et al., 2025), and reliability falls across multi-turn interactions (Laban et al., 2025). Existing studies largely remain within a single task, and standard agent benchmarks reset the session between tasks (Jimenez et al., 2024; Froger et al., 2026; Deng et al., 2025). How agents behave across sequences of tasks in a persistent session remains understudied.

Recent work studies ultra-long-context agents equipped with specialized memory systems, skill libraries, or curricula (He et al., 2026; Wang et al., 2026; Yao et al., 2026) (see Section 2). We ask a more basic question: can agents without such mechanisms reliably operate as state accumulates across tasks, and can training improve this capability?

To answer these questions, we construct sequential variants of GAIA2 (Froger et al., 2026) and SWE-Bench Pro (Deng et al., 2025). In GAIA2, agents complete up to 20 tasks while conversation history, memory, files, and environment state persist. In SWE-Bench Pro, agents process up to 40 issues from the same repository in commit order; the repository is restored to each issue's starting commit, while conversation history and the agent's home directory persist. Tasks and graders remain unchanged, and sequences are constructed so that intended solutions remain jointly feasible. This enables a direct comparison between independent and sequential evaluation.

We find that persistence has a model-dependent effect on task success, but consistently changes agent behavior. Across Opus 4.6, GPT-5.4, and DeepSeek v4 Pro, Opus 4.6 and GPT-5.4 remain near their independent baselines, whereas DeepSeek v4 Pro loses 5.9 percentage points on average. Across both benchmarks, all three models use up to 60% fewer output tokens and perform less exploratory work; on GAIA2, Python's share of tool calls increases by up to $4\times$. Trace analyses reveal both useful reuse and failure modes associated with reduced exploration and unchecked assumptions carried across tasks. Together, these findings suggest that managing accumulated context is a distinct model capability that training can improve.

To train this capability, we introduce Sequential Reinforcement Learning (Seq-RL) and apply it to Muse Glimmer-30B using persistent multi-task rollouts with local and downstream rewards. Trained only on disjoint GAIA2-style data, Seq-RL substantially narrows the sequential gaps on both GAIA2 splits and transfers zero-shot to SWE-Bench Pro. There, it reduces the $\mathrm{avg@3}$ gap to 6.0 percentage points, compared with 21.9 points for data-matched single-task RL, despite seeing no software-engineering data during RL. It also improves long-context retrieval and memory use.

Contributions. We contribute: **sequential evaluation**, a controlled protocol and two benchmark adaptations for persistent sequences of independently graded tasks; **behavioral characterization**, evidence that persistence changes effort, exploration, tool use, and context management even when average success remains stable; and **Seq-RL**, a training method that improves persistent-session performance and transfers across domains and agent harnesses.

2 RELATED WORK

Long-context evaluations show that performance declines with prompt length (Liu et al., 2023; Du et al., 2025), while multi-turn interactions further reduce reliability as user requirements accumulate (Laban et al., 2025). Agentic studies report related failures over long interaction histories (Chung et al., 2025; Fang et al., 2026; Sobal et al., 2026). SelfCompact shows that agents can adapt when to compact using a training-free rubric (Li et al., 2026a). These works study context accumulated within a single task or evolving objective. We instead study sequences of independently specified tasks, where agents must decide what state to reuse, revalidate, or ignore.

Most agent benchmarks reset the agent between tasks (Yao et al., 2024; Mialon et al., 2023; Jimenez et al., 2024; Sun et al., 2026). Recent software-engineering benchmarks instead construct dependent software engineering task sequences, exposing how errors propagate through an evolving codebase (Deng et al., 2026; Lam et al., 2026; Le et al., 2026; Shastry et al., 2026; Chen et al., 2026a). Siu et al. (2026) introduce ChainWorld, a closely related work. It composes OSWorld atomic tasks into

compatible chains and studies the effect of giving tasks all at once or in chains to the agent. We convert existing benchmarks (GAIA2 (Froger et al., 2026) and SWE-Bench Pro (Deng et al., 2025)) into sequences of independently scored tasks whose agent state systematically carries forward.

Memory and continual-learning benchmarks construct sequences specifically to test whether agents can retain and exploit relevant experience (Zheng et al., 2025; He et al., 2026; Shu et al., 2026; Asawa et al., 2026; Jiang et al., 2026). EvaLearn evaluates learning across related problem sequences, including settings with rubrics or teacher feedback (Dou et al., 2025), while ContinualSkillBench studies skill-linked tasks with feedback and explicit skill maintenance (Guan et al., 2026). Sequential embodied QA similarly carries memory across questions but focuses on memory architecture (Cai et al., 2026). Our sequences are not constructed to require cross-task reuse, they preserve ordinary agent state across existing benchmark tasks to isolate session persistence itself.

Recent training methods target cross-task reuse through generated tools, workflow memories, or skill libraries (Cai et al., 2024; Wang et al., 2025; 2026); progressive task specification (Chen et al., 2026b); explicit context updates (Chen et al., 2026c); or ordered curricula (Yao et al., 2026). CompactionRL jointly optimizes task execution and within-task compaction (Li et al., 2026b), while Learning to Remember trains memory updates using rewards from downstream queries (Zhang et al., 2026). Seq-RL instead introduces no dedicated memory architecture, skill representation, or hand-designed reuse mechanism: it trains the base agent directly on persistent task sequences, allowing cross-task behavior to emerge from optimization over local and downstream outcomes.

3 Experimental setup

In **independent evaluation**, each task starts in a fresh session with no prior conversation or home filesystem state. In **sequential evaluation**, each sequence starts clean, but its tasks share one session whose agent state persists. This comparison measures the end-to-end effect of session persistence under each benchmark's state semantics, while keeping task definitions and grading unchanged (see Figure 1, with more implementation details in Appendix Figure 8).

Benchmarks. Unlike EvaLearn and CL-Bench (Dou et al., 2025; Asawa et al., 2026), which are designed around sequential learning, we reuse independently defined and graded tasks grouped by a shared environment: a universe in GAIA2 or a repository in SWE-Bench Pro. **GAIA2.** The benchmark evaluates assistants in a simulated smartphone environment containing email, messaging, calendar, and contact applications (Froger et al., 2026). We evaluate the *Search* (160 tasks) and *Execution* (95 tasks) splits, organized into 20 sequences of 4–20 tasks. Tasks are grouped into *universes*, each representing a self-contained user environment with shared application state. Tasks are ordered for causal compatibility: completing an earlier task as intended does not invalidate a later one. Each sequence runs in one OpenClaw (OpenClaw Team, 2026) session, with both agent and environment state persisting. **SWE-Bench Pro.** The benchmark contains GitHub issues from 11 repositories (Deng et al., 2025). We group issues by repository, order them by Git history, and retain up to 40 per repository, yielding 420 tasks. Each sequence runs in one Mini-SWE-Agent (Yang et al., 2024) session. Before each task, the repository is checked out to that task's starting commit, and the agent is notified. Conversation and agent-home state persist. We make minimal extensions to Mini-SWE-Agent with persistent sessions and context compaction (Section A.2). Further task-bundling details, sequence-length distributions, and harness modifications in Appendix A.

Evaluation protocol. We evaluate GPT-5.4, Opus 4.6, and DeepSeek v4 Pro under both evaluation modes, using $K = 5$ rollouts per GAIA2 scenario and $K = 3$ per SWE-Bench Pro task. Each sequential rollout is initialized independently and executes a complete sequence end to end. Both harnesses use a 200K-token context window and all models use high reasoning effort. Infrastructure failures (e.g., VM crashes or model API outages) are replayed so the failed attempt does not affect later tasks.

Metrics. For each task, we collect K rollouts under both modes. Our primary metric is $\mathrm{avg@}K$, the mean success rate across rollouts. We report $\mathrm{pass@}K$ (coverage) and pass^K (all-run success) as secondary metrics in Appendix. For each metric m, the sequential gain is $\Delta_m = m_{\mathrm{seq}} - m_{\mathrm{ind}}$, where negative values indicate degradation under sequential evaluation.

4 SEQUENTIAL EVALUATION RESULTS

We first examine how session persistence affects task performance. We then analyze how agents adapt their behavior and tool use strategies as experience accumulates.

4.1 TASK PERFORMANCE UNDER PERSISTENCE

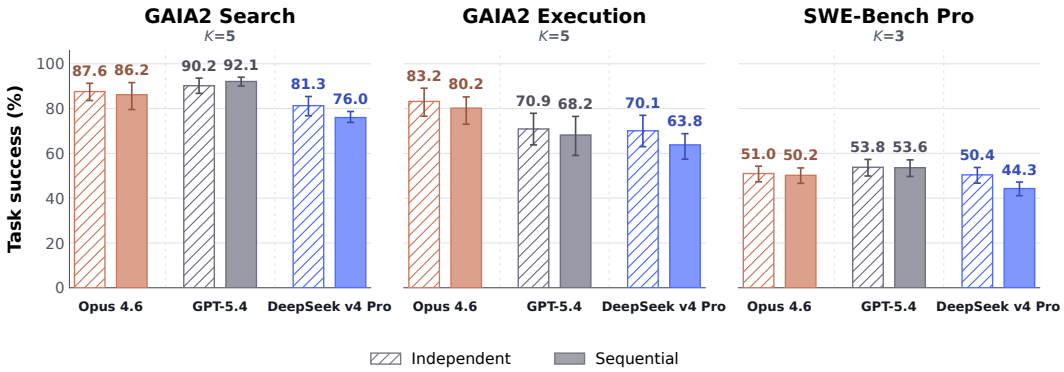


Figure 2: **Independent versus sequential mean task success.** Values above bars show avg@K and error bars show sequence-sampled cluster-bootstrap 95% CIs.

Sequential evaluation gives models prior context that they can reuse, while also requiring them to operate over an increasingly long context. As shown in Figure 2, persistence has no uniform effect on mean task success. Among frontier closed-source models, Opus 4.6 and GPT-5.4 remain close to their independent baselines, with average sequential deltas of -1.7 and -0.3 pp across the three evaluation settings, respectively. By contrast, DeepSeek v4 Pro degrades in every setting, losing 5.3 pp on GAIA2 Search, 6.3 pp on GAIA2 Execution, and 6.1 pp on SWE-Bench Pro. As a complementary reliability analysis, we report pass@K and pass^K metrics in Table 2 and Table 3. pass@K changes little for all models, indicating that the fraction of tasks solved at least once remains similar under sequential evaluation. Because pass^K requires success in all K runs and depends nonlinearly on per-run success, it is expected to decline faster than avg@K when average performance decreases. For DeepSeek v4 Pro, the fraction of tasks solved in every run is substantially reduced, suggesting greater fragility in production settings where the model operates over long contexts and chains of tasks within a persistent session. DeepSeek v4 Pro degrades across all three evaluations, spanning two domains and agent harnesses, suggesting that this behavior is model-dependent.

Context-window pressure and compaction. At 200K, GAIA2 never triggers OpenClaw's full-history summarization, although tool outputs are compacted frequently (Section B.3). Coding independent-to-sequential outcome changes as regressions (-1), stable outcomes (0), or recoveries ($+1$), we find that compaction count is negatively associated with outcome change for all models and significantly so for GPT-5.4 ($\rho = -0.140$, $p < 0.001$) and DeepSeek v4 Pro ($\rho = -0.072$, $p = 0.012$). For GPT-5.4, regressions average 33.6 compactions, compared with 15.0 for stable outcomes and 10.7 for recoveries. These correlations do not establish that compaction causes regressions: difficult trajectories may generate more tool output, leading both to more compactions and worse outcomes. We also find no clear relationship between a task's position in the sequence and its performance change, arguing against monotonically increasing context pressure as a sufficient explanation. SWE-Bench Pro provides a useful contrast. Its longer sequences trigger repeated full-history summarization (Figure 14), yet compactions are not correlated with performance changes. One possible explanation is that the repository is reset between tasks and successive issues often concern different parts of the codebase, making older tool observations less relevant. Compaction may therefore matter when it removes information that remains useful across tasks, but it does not provide a general explanation for the effects of sequential evaluation. To understand what changes beyond the final outcome, we next analyze agent behavior change.

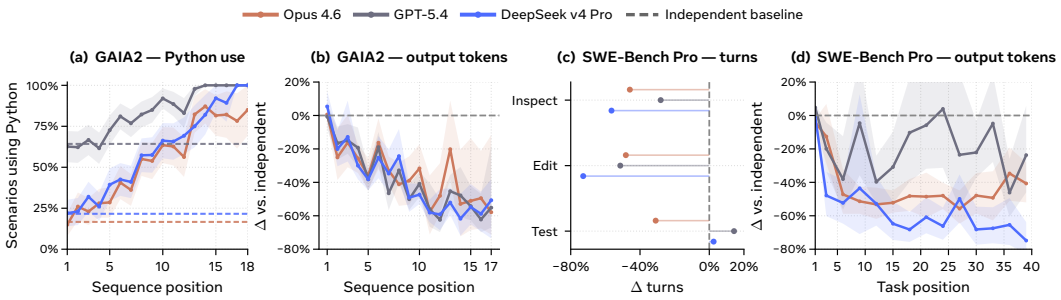


Figure 3: **Behavioral adaptation under sequential evaluation.** (a) GAIA2 Python use by sequence position; (b–d) Changes relative to independent evaluation in GAIA2 output tokens (b), SWE-Bench Pro turns by action type (c), and SWE-Bench Pro output tokens (d).

4.2 Behavioral changes under sequential evaluation

Shared behavioral change. Under sequential evaluation, all models follow shorter trajectories on both benchmarks. Sequential runs use 26 to 51% fewer LLM turns and 20 to 60% fewer output tokens than independent runs, with the reductions generally increasing over sequence position (Figure 3(b,d); Section B.2). Because each task is compared with its own independent run, the trend reflects a change in behavior, not easier later tasks. These reductions measure output effort rather than total inference cost, which may increase as sequential runs process longer input contexts. Shorter trajectories can reflect genuine reuse of prior work, or simply less effort on later tasks.

GAIA2: Python adoption. On GAIA2, shorter trajectories coincide with increasing Python use across all three models (Figure 3(a,b)). Trace inspection reveals two recurring uses: agents learn to orchestrate existing tools, and to extend those tools by implementing missing features. We provide examples of both behaviors in Section B.2. Because these scripts remain available on disk across tasks, they form a reusable tool layer. Their value, however, depends on the assumptions encoded in the code. One recurrent failure is incomplete pagination: the example in the appendix assumes that retrieving 130 contacts is sufficient. Across 1,574 sequential scenario runs from the three models, scripts using such a fixed bound succeed 9.5 pp less often than scripts that verify complete retrieval. Although observational, this comparison illustrates the central trade-off of persistent tools: they can preserve useful procedures, but also carry unchecked assumptions into later tasks.

SWE-Bench Pro: adapting across repo history. The repository advances through progressively newer commits, giving persistent agents an opportunity to accumulate knowledge of the codebase architecture, conventions, and test behavior. Across all three models, this accumulated context is associated with a more direct workflow: inspection turns fall by 28–56%, agents reach their first edit 1.8–9.0 turns sooner, and they inspect 24–32% fewer files (Figure 3(c), Section B.2). For Opus 4.6 and GPT-5.4, this effort reduction leaves avg@3 within 0.8 and 0.2 pp of their independent baselines, respectively. DeepSeek v4 Pro reduces output tokens and turns more aggressively, but its avg@3 falls by 6.1 pp. Direct analysis of its traces shows a tendency to commit early to broad, poorly localized changes and subsequently rewrite existing tests to accommodate those changes. Across matched evaluations, it modifies existing test files $10\times$ more sequentially than independently. Thus, accumulated context can support efficient navigation of a familiar codebase, but it can also encourage premature commitment when prior assumptions replace localized investigation and verification.

Takeaway. To test whether context-window pressure drives these effects, we do an ablation on GAIA2 at 32K, forcing more frequent compaction, and on SWE-Bench Pro at 1M, providing more headroom for accumulated context (Section B.3). Neither restores sequential performance across models: results remain model-dependent at 32K, while performance falls further for all models at 1M. Together, these results suggest that robustness depends on how agents use and manage accumulated context, motivating training directly on persistent task sequences.

5 SEQ-RL: TRAINING FOR PERSISTENT SESSIONS

Typical agent RL trains on isolated tasks from fresh context and therefore does not expose models to trajectory-dependent histories, though recent work targets long-lifecycle or cross-task learning (Chen et al., 2026c; Yao et al., 2026). We introduce *Seq-RL* and train Muse Glimmer-30B[1] directly in persistent multi-task sessions.

5.1 SEQUENTIAL REINFORCEMENT LEARNING SETUP

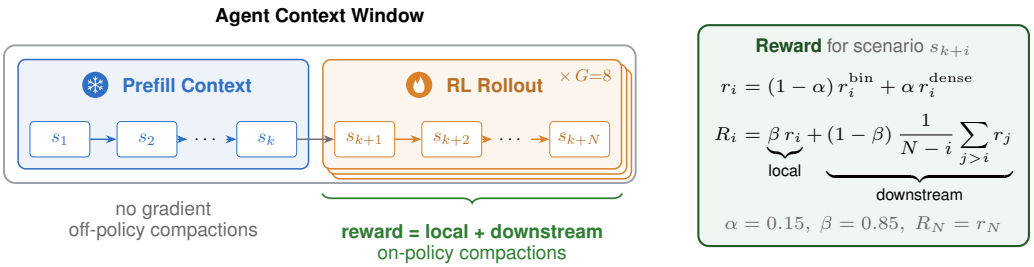


Figure 4: **Sequential reinforcement learning.** ❄ The frozen prefix receives no gradient. 🔥 The policy samples $G = 8$ continuations of the same next $N = 6$ scenarios and is trained on them.

Training data. We annotate 27K GAIA2-style scenarios from universes disjoint from the evaluation set and chain them into 1.6K compatible sequences of up to 50 scenarios (Section C.1). Each sequence yields one training example: a frozen prefix followed by $N = 6$ trainable scenarios. The resulting 9.6K trainable scenarios are the only ones that receive gradient; the other scenarios serve only as prefix context or are unused.

Seq-RL rollout construction. To build a prefix, the initial Muse Glimmer-30B checkpoint runs the sequence from its first scenario until it has compacted at least c times, with c sampled from $\{0, 1, 2\}$, and its context exceeds about 100K tokens; it then stops at the next scenario boundary s_k, where we snapshot the complete agent and environment state. Prefixes are generated once and stay fixed, becoming off-policy as training proceeds; about 25% of prefix scenarios are solved, so snapshots contain failures and compaction artifacts as well as successes. From each snapshot, the policy samples $G = 8$ independent continuations over the next six scenarios, $s_{k+1}, \ldots, s_{k+6}$, without task-level resets (Figure 4). Only model-generated tokens in these scenarios, including compactions, receive gradient; prefix tokens and tool or environment observations are masked.

Reward and optimization. Each scenario's local reward r_i mixes binary task success with a dense F_2 score over its write actions ($\alpha = 0.15$). Its training reward R_i mixes r_i with the mean local reward of the later scenarios in the same continuation (downstream credit; $\beta = 0.85$; $R_N = r_N$). Averaging rather than summing keeps R_i on the same scale at every position and fixes the local/downstream split at 85/15. Every trainable token generated during scenario i, including compaction tokens, receives R_i. We use a critic-free, GRPO-style objective (Shao et al., 2024) with one baseline per prefix group: the token-weighted mean reward over all positions of its $G = 8$ continuations (Section C.1).

Training configuration. Each step samples $B = 32$ prefix groups, i.e. 256 six-scenario continuations, with full-model training at a 128K-token context window; remaining settings are in Section C.1.

Controls. Two controls isolate what Seq-RL adds. *Single-task RL* (abbreviated RL) is the same recipe with no prefix and $N = 1$: each trainable scenario is sampled from a fresh session and receives only its local reward, which tests whether the data alone explains Seq-RL's gains. *Seq-RL with $\beta = 1$* keeps the prefixes and sequences but drops downstream credit, isolating its contribution (Section C.3). All runs are data-matched: they start from the same checkpoint and see exactly the same 9.6K scenarios, each sampled $G = 8$ times per epoch for three epochs.

[1]Model weights downloaded from HuggingFace

5.2 SAME DATA, MORE SIGNAL

For a model already post-trained on tool use and agentic environments, the 9.6K scenarios offer little to learn in isolation: single-task RL's mean training reward rises by only 3%, from 0.537 to 0.555 (Figure 15). Presented in sequences, after a 100K-token prefix, the same scenarios become a strong training signal: Seq-RL's mean training reward rises by 91%, from 0.232 to 0.443. The signal thus comes from how the data is presented, not from the data itself.

We track how the sequential gap Δpass^5 evolves during training on held-out GAIA2 (Figure 5). We use pass^5 because it counts a scenario only when all five rollouts succeed, so it tracks a lower bound on what the model has learned. Seq-RL narrows the gap at every checkpoint, from -12.5 to -3.1 pp on Search and from -6.3 to -1.0 pp on Execution, while improving both independent and sequential pass^5 and avg@5 (Table 9), so the narrowing is not caused by declining independent performance. Single-task RL leaves the Search gap unchanged and widens the Execution gap to -16.8 pp: its independent gains (pass^5 66.3 to 74.7) do not carry into persistent sessions (60.0 to 57.9). Downstream credit adds a modest further benefit (Section C.3).

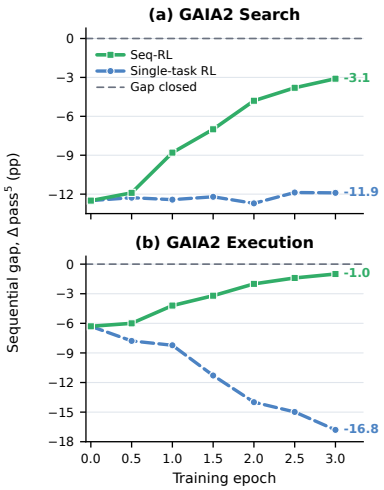


Figure 5: **Seq-RL narrows the sequential gap during training.** We report $\Delta\text{pass}^5 = \text{pass}^5_{\text{seq}} - \text{pass}^5_{\text{ind}}$.

5.3 SEQ-RL IMPROVES AGENTS UNDER PERSISTENT CONTEXT

Gains carry over to persistent sessions. Figure 6 compares the base Muse Glimmer-30B with its single-task RL and Seq-RL checkpoints on GAIA2 (avg@5) and SWE-Bench Pro (avg@3); Table 9 reports all metrics. Seq-RL is the only trained checkpoint whose gains carry over to persistent sessions: it has the best sequential avg@K and pass^K on all three evaluations while remaining competitive independently. Single-task RL instead improves only in isolation: on GAIA2 Execution, it ranks first independently (88.6) but last sequentially (82.1). On GAIA2, Muse Glimmer-30B is already near its ceiling in isolation (91.4 and 86.1 avg@5) and loses only a few points sequentially, so the sequential avg@5 gaps leave little room to improve (Section 5.2).

The benefit transfers zero-shot to software engineering. SWE-Bench Pro tests transfer across both domain and harness: the RL data contains no software-engineering tasks, and evaluation uses

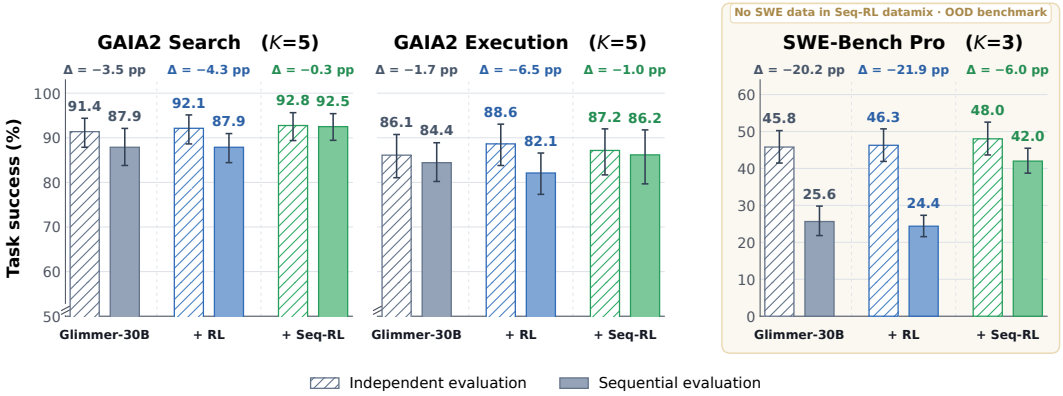


Figure 6: **Seq-RL narrows the sequential gap.** Avg@5 on GAIA2 Search and Execution and avg@3 on SWE-Bench Pro with a 200K-token context window. Error bars show 95% clustered-bootstrap CIs obtained by resampling tasks or complete sequential sessions, rather than individual generations.

Mini-SWE-Agent rather than OpenClaw. Muse Glimmer-30B drops from 45.8 avg@3 independently to 25.6 sequentially, a -20.2 pp gap well beyond any frontier model's in Section 4 (at worst -6.3 pp), possibly because it is a much smaller dense model. Single-task RL leaves both scores essentially unchanged (46.3 and 24.4, within the 95% CIs). Seq-RL raises sequential avg@3 to 42.0 while independent avg@3 barely moves (48.0), shrinking the gap to -6.0 pp; pass@3 and pass^3 improve as well (Table 9). The learned behavior is thus neither specific to GAIA2 nor a gain in isolated-task ability. After Seq-RL, all three of the model's gaps (-0.3, -1.0, and -6.0 pp) fall within the frontier range.

Seq-RL changes agent behavior. Following Section 4.2, we examine whether these gains coincide with changes in agent behavior. On GAIA2, Seq-RL invokes Python more often than the other checkpoints in both modes, for example in 80.3% of independent and 96.1% of sequential Search scenarios, versus 33.6% and 67.0% for Muse Glimmer-30B (Figure 7a; Table 10). Qualitative inspection suggests that the model uses Python as an external workspace, recomputing task-specific evidence rather than trusting information carried in context. A related change appears on SWE-Bench Pro. For Muse Glimmer-30B and single-task RL, resolve rate declines with sequence position, empty-patch submissions increase (accounting for about 30% of the drop in resolve rate), and testing approaches zero (Figure 7b–d). Seq-RL largely avoids all three trends: it averages 2.7 test turns per task sequentially, compared with 1.1–1.2 for the other checkpoints, while still using about 54% fewer output tokens than in independent evaluation, and it also tests more independently (4.4 vs. 3.5–3.8 test turns; Table 11). These results suggest that Seq-RL shifts the policy toward refreshing evidence and verifying outputs in both modes, which matters most when accumulated context invites skipping that work.

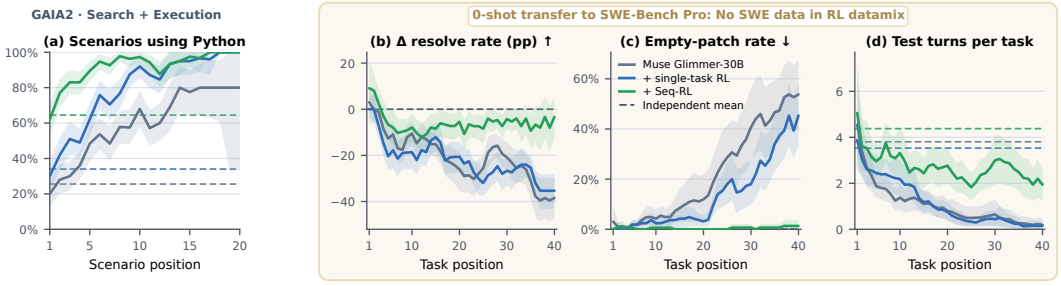


Figure 7: **Seq-RL sustains effort across task sequences.** (a) On GAIA2, Seq-RL uses Python in more scenarios throughout the sequence. (b–d) On SWE-Bench Pro, it limits the position-dependent decline in resolve rate, avoids empty patches, and keeps testing. Dashed lines show independent means; SWE-Bench Pro curves use five-position rolling means; bands show 95% clustered-bootstrap CIs.

5.4 Transfer beyond sequential evaluation

Seq-RL trains the model to find and reuse information in a 100K-token history. We test whether this transfers beyond sequential evaluation, without further training, on two held-out benchmarks (Table 1): MRCR v2 (Vodrahalli et al., 2024), a non-agentic long-context retrieval task, and MemoryArena (He et al., 2026), which tests memory use across interdependent sessions in new domains. We use each benchmark's official evaluation code. On MRCR v2, both RL checkpoints improve over Muse Glimmer-30B, and Seq-RL scores highest at every context length (36.2, 27.0, and 22.3 at 32K, 64K, and 128K), although its intervals overlap with those of single-task RL. MemoryArena separates the two more clearly: single-task RL changes little, improving travel-planning progress (3.1 to 4.9) but lowering formal-physics scores, whereas Seq-RL improves progress in all four domains, most in formal physics (60.8 to 71.4 progress, 35.0 to 55.0 success rate) and travel planning (3.1 to 10.3 progress). These results suggest that Seq-RL strengthens capabilities, long-context retrieval and memory use, that are reusable beyond sequential task execution.

Table 1: **Out-of-distribution benchmark generalization.** Entries report the mean with the 95% CI shown in smaller type. MRCR v2 uses avg@4 for eight-needle retrieval; MemoryArena uses avg@5 over five runs for Progress Score (PS) and Success Rate (SR).

Method	**MRCR v2** (8 needles)			**MemoryArena** (concatenated history)							
	avg@4↑			**Formal math**		**Formal physics**		**Bundled Shopping**		**Travel Planning**	
	32K	**64K**	**128K**	**PS**	**SR**	**PS**	**SR**	**PS**	**SR**	**PS**	**SR**
Baseline	26.3 ± 3.0	19.0 ± 2.7	13.0 ± 2.2	34.9 ± 1.1	2.0 ± 1.4	60.8 ± 1.0	35.0 ± 4.4	50.1 ± 0.8	4.0 ± 0.6	3.1 ± 0.4	0.1 ± 0.2
RL	34.7 ± 3.5	23.8 ± 2.9	19.5 ± 2.1	35.1 ± 0.8	$\mathbf{3.0} \pm 1.4$	58.4 ± 1.0	25.0 ± 4.4	50.0 ± 0.8	4.0 ± 0.6	4.9 ± 0.5	0.4 ± 0.3
Seq-RL	$\mathbf{36.2} \pm 2.4$	$\mathbf{27.0} \pm 3.7$	$\mathbf{22.3} \pm 2.9$	$\mathbf{36.9} \pm 0.8$	$\mathbf{3.0} \pm 1.4$	$\mathbf{71.4} \pm 1.5$	$\mathbf{55.0} \pm 4.4$	$\mathbf{52.9} \pm 0.8$	$\mathbf{5.3} \pm 0.6$	$\mathbf{10.3} \pm 0.5$	$\mathbf{0.7} \pm 0.3$

6 Discussion

Limitations. We measure the end-to-end sequential gap but do not systematically decompose its root causes (environment corruption, information loss, faulty compaction, stale beliefs, reduced effort, etc.). Isolating these effects would require a dedicated study. Prior work separately documents multi-turn degradation, lossy compaction, and error propagation from retained experience (Laban et al., 2025; Li et al., 2026a; Xiong et al., 2026). Sequence orders are fixed by causal compatibility in GAIA2 and by commit history in SWE-Bench Pro, which keeps sequences jointly feasible but prevents measuring sensitivity to alternative orders. More broadly, our tasks are independently specified, and longer-horizon sequences with stronger dependencies between tasks remain to be built and studied. On the training side, we run one seed per experiment given long-context RL is computationally expensive. Further work should explore Seq-RL's design choices (prefill, reward function, sequence lengths, etc.) and test more efficient alternatives.

A training–deployment gap. Sequential evaluation tests agents through sustained interactions containing prior actions, tool outputs, failures, summaries, and agent-created files. Agents must decide what to reuse, verify, or ignore, making this setting closer to how coding and personal agents are deployed than standard independent evaluation. It can also expose weaknesses in benchmarks and agent infrastructure because a faulty tool, grader loophole, or corrupted state can affect several later tasks. Related work explicitly benchmarks such shortcut and evaluator-exploitation behavior under independent and chained tool-use regimes (Thaman, 2026). For example, in an excluded pilot evaluation with a misconfigured SWE-Bench Pro environment, sequential agents learned from earlier tasks to access hidden commit history or retrieve gold patches, reaching 100% success by the third task. This behavior did not emerge under independent evaluation, where the relevant experience was discarded after each task. This illustrates how sequential evaluation can uncover vulnerabilities that isolated episodes may miss. We believe sequential evaluation and training offer a promising path toward agents that operate reliably over ultra-long interactions and make better use of compaction, memory, and in-context learning.

7 Conclusion

We introduced sequential evaluation, a simple protocol that removes task-level resets while leaving tasks and grading unchanged. Across GAIA2 and SWE-Bench Pro, persistence had model-dependent effects on success and substantially changed how agents allocated effort, even when average accuracy remained stable. Agents reused knowledge and tools to shorten their trajectories, but could also propagate stale state or take harmful shortcuts. Training on persistent multi-task rollouts with Seq-RL narrowed these sequential gaps and transferred beyond its training domain. More broadly, agents are trained and evaluated one task at a time but deployed in persistent sessions. Sequential evaluation applies to any benchmark whose tasks share an environment, and offers a natural setup to train and evaluate in-context learning, as agents learn their environment and users over time. Overall, reliable long-lived agents must not only solve individual tasks, but also manage their accumulated experience.

AI USE STATEMENT

We used Anthropic Claude Code and OpenAI Codex as coding assistants during implementation and experimentation. They helped us modify the codebase, write plotting scripts, prepare figures, search recent submissions for related work, and make occasional wording edits. After running deterministic compatibility checks, we also used Claude Code to flag possible semantic conflicts in the ordering of training and GAIA2 scenarios. All 27K training scenarios were labeled by humans and were not generated using these tools. We did not use generative AI to design Seq-RL or the experiments, or to analyze or interpret the results. AI was used to get feedback on earlier draft version of the paper but not to explore new research ideas.

We reviewed every AI-assisted change before using it. We tested the code, compared the figures with the underlying results, checked citations against the original papers, and reviewed all manuscript edits. We also manually examined each scenario-ordering conflict flagged by Claude Code. We remain responsible for the paper and all accompanying artifacts.

ETHICS STATEMENT

Our experiments use simulated user environments and publicly available software repositories included in SWE-Bench Pro. They do not involve human-subject experiments or deployment to real users or production systems. Persistent agents may retain incorrect or sensitive information and propagate harmful actions across tasks. Sequential evaluation is intended to expose these risks and support the development of safer, more reliable long-lived agents. All experiments were conducted in isolated benchmark environments with no internet access.

REPRODUCIBILITY STATEMENT

Section 3 describes the evaluation protocol, model settings, metrics, and treatment of infrastructure failures. Section 5.1 presents the Seq-RL method, while Appendix A documents sequence construction and the modifications to Mini-SWE-Agent. Appendix C reports the training configuration, compute requirements, ablations, and complete evaluation results. We will release evaluation code and training data upon publication.

REFERENCES

Parth Asawa, Christopher M. Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E. Gonzalez. Continual learning bench: Evaluating frontier AI systems in real-world stateful environments, 2026. URL https://arxiv.org/abs/2606.05661.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=qV83K9d5WB.

Zikui Cai, Kaushal Janga, Tan Dat Dao, Seungjae Lee, Shivin Dass, Mingyo Seo, Kaiyu Yue, Mintong Kang, Nandhu Pillai, Monte Hoover, Aadi Palnitkar, Ruchit Rawal, Ruijie Zheng, Bo Li, Yuke Zhu, Roberto Martín-Martín, Tom Goldstein, and Furong Huang. Beyond episodic evaluation: Memory architectural bottlenecks in sequential embodied question answering, 2026. URL https://arxiv.org/abs/2607.21571.

Jialong Chen, Xander Xu, Hu Wei, Chuan Chen, and Bing Zhao. SWE-CI: Evaluating agent capabilities in maintaining codebases via continuous integration, 2026a. URL https://arxiv.org/abs/2603.03823.

Tianlang Chen, Shirley Wu, and Jure Leskovec. Found in conversation: LLMs teach themselves to close the multi-turn gap, 2026b. URL https://arxiv.org/abs/2605.24432.

Yanxi Chen, Weijie Shi, Yuexiang Xie, Boyi Hu, Yaliang Li, Bolin Ding, and Jingren Zhou. Connect the dots: Training LLMs for long-lifecycle agents with cross-domain generalization via reinforcement learning, 2026c. URL https://arxiv.org/abs/2606.20002.

Andy Chung, Yichi Zhang, Kaixiang Lin, Aditya Rawal, Qiaozi Gao, and Joyce Chai. Evaluating long-context reasoning in LLM-based WebAgents, 2025. URL https://arxiv.org/abs/2512.04307.

Gangda Deng, Zhaoling Chen, Zhongming Yu, Haoyang Fan, Yuhong Liu, Yuxin Yang, Dhruv Parikh, Rajgopal Kannan, Le Cong, Mengdi Wang, Qian Zhang, Viktor Prasanna, Xiangru Tang, and Xingyao Wang. SWE-Milestone: Evaluating AI agents on continuous software evolution, 2026. URL https://arxiv.org/abs/2603.13428.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks?, 2025. URL https://arxiv.org/abs/2509.16941.

Shihan Dou, Ming Zhang, Chenhao Huang, Jiayi Chen, Feng Chen, Shichun Liu, Yan Liu, Chenxiao Liu, Cheng Zhong, Zongzhang Zhang, Tao Gui, Chao Xin, Chengzhi Wei, Lin Yan, Yonghui Wu, Qi Zhang, and Xuanjing Huang. EvaLearn: Quantifying the learning capability and efficiency of LLMs via sequential problem solving, 2025. URL https://arxiv.org/abs/2506.02672.

Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. Context length alone hurts LLM performance despite perfect retrieval, 2025. URL https://arxiv.org/abs/2510.05381.

Shicheng Fang, Yuxin Wang, Xiaoran Liu, Jiahao Lu, Chuanyuan Tan, Xinchi Chen, Yining Zheng, Xuanjing Huang, and Xipeng Qiu. AgentLongBench: A controllable long benchmark for long-contexts agents via environment rollouts, 2026. URL https://arxiv.org/abs/2601.20730.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Gaia2: Benchmarking LLM agents on

dynamic and asynchronous environments, 2026. URL https://arxiv.org/abs/2602.11964.

Tianyi Guan, Yiding Wang, Haotong Yang, Siyuan Cao, Shirui Liu, Yi Hu, Jiaqi Li, and Muhan Zhang. ContinualSkillBench: Can LLM agents truly evolve their capabilities?, 2026. URL https://arxiv.org/abs/2608.03874.

Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, Jiaxin Pei, Julian McAuley, Yejin Choi, and Alex Pentland. MemoryArena: Benchmarking agent memory in interdependent multi-session agentic tasks, 2026. URL https://arxiv.org/abs/2602.16313.

Sihang Jiang, Lipeng Ma, Zhonghua Hong, Keyi Wang, Zhiyu Lu, Tengfei Wang, Shisong Chen, Jinghao Zhang, Tianjun Pan, Weijia Li, Jiaqing Liang, and Yanghua Xiao. SEA-Eval: A benchmark for evaluating self-evolving agents beyond episodic assessment, 2026. URL https://arxiv.org/abs/2604.08988.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation, 2025. URL https://arxiv.org/abs/2505.06120.

Man Ho Lam, Chaozheng Wang, Hange Liu, Jingyu Xiao, Haau sing Li, Jen tse Huang, Terry Yue Zhuo, and Michael R. Lyu. SWE-Chain: Benchmarking coding agents on chained release-level package upgrades, 2026. URL https://arxiv.org/abs/2605.14415.

Tue Le, Minh V. T. Thai, Dung Nguyen Manh, Huy Phan Nhat, and Nghi D. Q. Bui. SWE-EVO: Benchmarking coding agents in long-horizon software evolution scenarios, 2026. URL https://arxiv.org/abs/2512.18470.

Tianjian Li, Jingyu Zhang, William Jurayj, Xi Wang, Chuanyang Jin, Mehrdad Farajtabar, Eric Nalisnick, and Daniel Khashabi. Self-compacting language model agents, 2026a. URL https://arxiv.org/abs/2606.23525.

Yujiang Li, Zhenyu Hou, Yi Jing, Jie Tang, and Yuxiao Dong. CompactionRL: Reinforcement learning with context compaction for long-horizon agents, 2026b. URL https://arxiv.org/abs/2607.05378.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023. URL https://arxiv.org/abs/2307.03172.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants, 2023. URL https://arxiv.org/abs/2311.12983.

Nous Research. Hermes Agent: A self-improving agent scaffold, 2026. URL https://github.com/NousResearch/hermes-agent.

OpenClaw Team. OpenClaw: An open-source agent scaffold, 2026. URL https://github.com/openclaw/openclaw. v2026.4.1.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

KN Ajay Shastry, Ganesh Senrayan, Shrey Satapara, Pranoy Panda, and Chaitanya Devaguptapu. Beyond isolated tasks: A framework for evaluating coding agents on sequential software evolution, 2026. URL https://arxiv.org/abs/2604.03035.

Yiheng Shu, Bernal Jiménez Gutiérrez, Saisri Padmaja Jonnalagedda, Yuguang Yao, Huan Sun, and Yu Su. AgentCL: Toward rigorous evaluation of continual learning in language agents, 2026. URL https://arxiv.org/abs/2606.02461.

Vincent Siu, Manasi Sharma, Dawn Song, Daniel Yue Zhang, Chenguang Wang, and Ying Liu. ChainWorld: Composing long-horizon desktop workloads from atomic OSWorld tasks, 2026. URL https://arxiv.org/abs/2606.21654.

Vlad Sobal, Shuo Yang, Yuting Zhang, Wei Xia, and Stefano Soatto. StaminaBench: Stress-testing coding agents over 100 interaction turns, 2026. URL https://arxiv.org/abs/2606.19613.

Yiyou Sun, Xinyang Han, Weichen Zhang, Yuanbo Pang, Tianyu Wang, Yuhan Cao, Yixiao Huang, Chris Duroiu, Haoyun Zhang, Jeffrey Lin, et al. Agents' last exam, 2026. URL https://arxiv.org/abs/2606.05405.

Kunvar Thaman. Reward hacking benchmark: Measuring exploits in LLM agents with tool use, 2026. URL https://arxiv.org/abs/2605.02964. Accepted to ICML 2026.

Kiran Vodrahalli, Santiago Ontanon, Nilesh Tripuraneni, Kelvin Xu, Sanil Jain, Rakesh Shivanna, Jeffrey Hui, Nishanth Dikkala, Mehran Kazemi, Bahare Fatemi, Rohan Anil, Ethan Dyer, Siamak Shakeri, Roopali Vij, Harsh Mehta, Vinay Ramasesh, Quoc Le, Ed Chi, Yifeng Lu, Orhan Firat, Angeliki Lazaridou, Jean-Baptiste Lespiau, Nithya Attaluri, and Kate Olszewska. Michelangelo: Long context evaluations beyond haystacks via latent structure queries, 2024. URL https://arxiv.org/abs/2409.12640.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library, 2026. URL https://arxiv.org/abs/2512.17102.

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. URL https://proceedings.mlr.press/v267/wang25bx.html.

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, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.27. URL https://aclanthology.org/2026.acl-long.27/.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Zhiyuan Yao, Yuxin Chen, Zhengxi Lu, Zishan Xu, Yueqing Sun, Yifu Guo, Yuquan Lu, Zhengzhou Cai, Kangning Zhang, Zhuowen Han, Zi-Han Wang, Ziang Ye, Qi Gu, Xunliang Cai, Weiwen Liu, and Yongliang Shen. SkillRise: Agentic reinforcement learning for cross-task skill evolution, 2026. URL https://arxiv.org/abs/2607.26784.

Kehao Zhang, Shangtong Gui, Sheng Yang, Wei Chen, and Yang Feng. Learning to remember: End-to-end training of memory agents for long-context reasoning, 2026. URL https://arxiv.org/abs/2602.18493.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. LifelongAgentBench: Evaluating LLM agents as lifelong learners, 2025. URL https://arxiv.org/abs/2505.11942.

Appendix

The appendix follows the structure of the main text: Appendix A gives experimental details, Appendix B reports additional results for sequential evaluation (Section 4), and Appendix C gives details and additional results for Seq-RL (Section 5).

A Experimental details

A.1 Building the sequential benchmarks

We convert GAIA2 and SWE-Bench Pro into sequential benchmarks by removing task-level session resets while keeping task definitions and grading unchanged. Tasks are grouped by their shared environment (a user universe in GAIA2, a Git repository in SWE-Bench Pro), and each group forms one persistent sequence. Figure 8 illustrates the protocol and Figure 9 the resulting sequence lengths.

GAIA2. Each universe and split defines one sequence. In the original benchmark, every scenario starts from the same initial universe state; sequentially, scenario $t + 1$ starts from the state left by scenario t. We therefore order scenarios so that completing one as intended never makes a later one infeasible. A deterministic checker uses GAIA2's write-action DAG annotations to detect conflicting state modifications, and a Claude Code review flags semantic conflicts that the annotations miss. Of the 160 Execution scenarios, only 95 can be arranged into such compatible sequences; we drop the remaining 65 and keep all 160 Search scenarios. Compatibility holds under the intended solutions only: the effects of erroneous agent actions are deliberately allowed to propagate. Neither the environment nor the OpenClaw session is reset, so conversation, memory, files, compactions, and environment changes persist throughout each sequence. Independent evaluation uses the same 255 scenarios, so every comparison is paired.

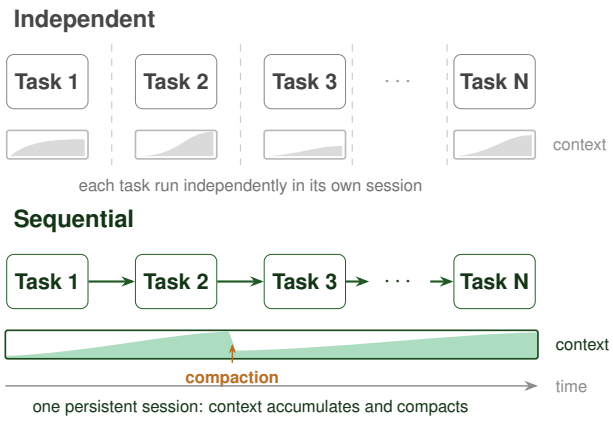


Figure 8: **Sequential evaluation setup.** Independent evaluation runs each task in a fresh session. Sequential evaluation processes independently specified tasks in one persistent session, carrying conversation and files across tasks as context accumulates and compacts. GAIA2 additionally preserves memory and environment state, whereas SWE-Bench Pro resets the repository before each task.

SWE-Bench Pro. Each repository defines one sequence. We order issues by their starting commits, so that tasks follow the evolution of the codebase, and keep the first 40 per repository, yielding 420 tasks in 11 sequences. Before each task, we clean the working tree, check out the task's starting commit, and notify the agent, so that it does not assume its previous patch is still applied. Agent-visible checkouts are scrubbed of Git history and artifacts that could reveal reference patches, and the agent has no internet access. In principle, the agent can still inspect how the repository evolved after an earlier issue; we did not observe agents recovering reference solutions this way. Only the repository is reset: conversation, compaction history, and the agent's home directory persist, as for a coding agent that moves between checkouts without restarting its session.

A.2 Harnesses

OpenClaw (GAIA2). We use OpenClaw's native context management, which has two mechanisms: tool-output compaction, which compacts old tool outputs, and full-history summarization, which

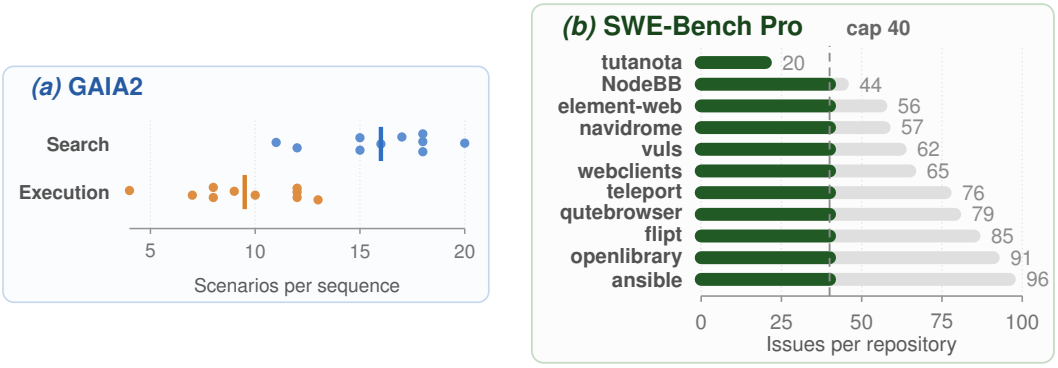


Figure 9: **Sequence lengths.** *(a)* GAIA2 scenarios per sequence for the Search and Execution splits: 255 tasks in 20 sequences of 4–20. *(b)* SWE-Bench Pro issues per repository; green marks the tasks retained in each sequence under the 40-task cap. The cap binds for ten of the eleven repositories, while `tutao/tutanota` contributes only 20, giving 420 tasks in total.

summarizes older history when the context nears the window limit. Table 6 reports how often each occurs.

Mini-SWE-Agent (SWE-Bench Pro). Mini-SWE-Agent (Yang et al., 2024) is designed for independent, single-task episodes: each task runs in a fresh container, and `DefaultAgent.run()` resets the conversation. We add three mechanisms for sequential evaluation; disabling them recovers the stock behavior, so both modes run the same agent.

- **Persistent conversation session.** A session object keeps the message history across tasks, so task $t{+}1$ continues the conversation of task t.
- **Context compaction.** When the conversation nears the window limit, older history is condensed into a structured summary of goals, progress, and next steps while recent turns are kept, mirroring OpenClaw's full-history summarization.
- **Agent-home persistence.** The agent's home directory persists across tasks, although the repository checkout is reset.

The agent has no dedicated cross-task memory module: beyond the conversation, persistence is limited to files it creates in its home directory.

A.3 Evaluation protocol and statistics

We run $K = 5$ rollouts per GAIA2 scenario and $K = 3$ per SWE-Bench Pro task in each mode; a sequential rollout runs a complete sequence from a clean state. Infrastructure failures, such as VM failures, lost connectivity, or SSH tunnel and podman exceptions, are replayed so that they do not propagate to later tasks in the sequence.

Metrics. $\text{avg@}K$ is the mean success rate over the K rollouts, $\text{pass@}K$ the fraction of tasks solved in at least one rollout, and pass^K the fraction solved in all K; we report differences as $\Delta = \text{seq} - \text{ind}$, so negative values indicate degradation. We use two Python measures: *Python share*, the fraction of tool calls that execute Python (Table 4), and *Python use*, the fraction of scenario runs that call Python at least once (Figure 3(a), Figure 7(a), and Table 10).

Statistics. We report 95% cluster-bootstrap confidence intervals for $\text{avg@}K$ only; $\text{pass@}K$ and pass^K are point estimates. The resampling unit is the task for independent results, the complete sequence for sequential results, and the matched sequence for Δ. For Section 4.1, we compute Spearman correlations between the number of tool-output compactions in a GAIA2 scenario and its outcome change from independent to sequential, coded as regression (-1), stable (0), or recovery ($+1$).

B SEQUENTIAL EVALUATION: ADDITIONAL RESULTS

This section reports the complete results behind Section 4. Results use a 200K-token window unless stated otherwise.

B.1 TASK PERFORMANCE

Table 2 and Table 3 report avg@K with 95% intervals, and pass@K and pass^K as point estimates, for both modes. In every setting, pass@K stays close to its independent value, which suggests that the drops in avg@K discussed in Section 4.1 come from tasks solved in fewer runs rather than from tasks no longer solved at all.

Table 2: **GAIA2 full metrics report at 200K ($K = 5$).** Values are percentages; $\Delta = \text{Seq.} - \text{Indep.}$ For avg@5, $\pm$ is the half-width of the 95% cluster-bootstrap interval (Section A.3); other metrics are point estimates. Bold marks the best value per metric and split.

		GAIA2 Search			GAIA2 Execution		
Metric	**Model**	**Indep.**	**Seq.**	**Δ**	**Indep.**	**Seq.**	**Δ**
avg@5	Opus 4.6	87.6 ± 3.81	86.2 ± 5.95	−1.4 ± 4.64	**83.2** ± 6.21	**80.2** ± 6.10	−3.0 ± 5.28
	GPT-5.4	**90.2** ± 3.40	**92.1** ± 1.93	**+1.9** ± 2.97	70.9 ± 7.05	68.2 ± 8.64	**−2.7** ± 4.31
	DeepSeek v4 Pro	81.3 ± 4.30	76.0 ± 2.43	−5.3 ± 3.75	70.1 ± 7.02	63.8 ± 5.64	−6.3 ± 7.45
pass@5	Opus 4.6	96.2	**98.8**	**+2.6**	**92.6**	**90.5**	−2.1
	GPT-5.4	**98.1**	96.9	−1.2	89.5	89.5	**0.0**
	DeepSeek v4 Pro	96.2	96.2	0.0	87.4	86.3	−1.1
pass^5	Opus 4.6	70.0	68.8	−1.2	**67.4**	**61.1**	**−6.3**
	GPT-5.4	**76.2**	**81.2**	**+5.0**	47.4	33.7	−13.7
	DeepSeek v4 Pro	58.8	42.5	−16.3	38.9	30.5	−8.4

Table 3: **SWE-Bench Pro full metrics report at 200K ($K = 3$).** Values are percentages; $\Delta = \text{Seq.} - \text{Indep.}$ For avg@3, $\pm$ is the half-width of the 95% cluster-bootstrap interval (Section A.3); other metrics are point estimates. Bold marks the best value per metric.

Metric	**Model**	**Indep.**	**Seq.**	**Δ**
avg@3	Opus 4.6	51.0 ± 3.5	50.2 ± 3.4	−0.8 ± 0.9
	GPT-5.4	**53.8** ± 3.9	**53.6** ± 3.7	**−0.2** ± 1.2
	DeepSeek v4 Pro	50.4 ± 3.5	44.3 ± 3.1	−6.1 ± 2.7
pass@3	Opus 4.6	56.2	57.4	**+1.2**
	GPT-5.4	**60.2**	**60.2**	0.0
	DeepSeek v4 Pro	59.0	56.4	−2.6
pass^3	Opus 4.6	44.0	**40.0**	**−4.0**
	GPT-5.4	**44.5**	36.9	−7.6
	DeepSeek v4 Pro	41.2	30.5	−10.7

B.2 BEHAVIORAL CHANGES

On GAIA2, sequential runs already use fewer turns and output tokens at the second scenario of a sequence (Figure 10), long before the context approaches the window limit (Section B.3), so the reduction reflects a response to prior context rather than context pressure. Python's share of tool calls rises with persistence, also rises when only the window shrinks, and is highest when both apply (Table 4): agents increasingly process tool output in scripts that keep raw records out of the context, without being prompted to (Figure 11). Because these scripts remain available to later tasks, their assumptions carry forward as well. Among 1,574 sequential scenario runs in which Python automates pagination, scripts whose termination logic assumes a fixed bound succeed in 77.2% of runs, against 86.7% for scripts that verify complete retrieval; these rates are pooled over models and tasks, unadjusted, and observational.

On SWE-Bench Pro, the shorter sequential trajectories come mostly from exploration: agents inspect fewer files and reach their first edit sooner, while the share of turns spent testing rises for every model (Table 5). Persistent agents thus skip re-exploring code they have already seen but keep verifying their

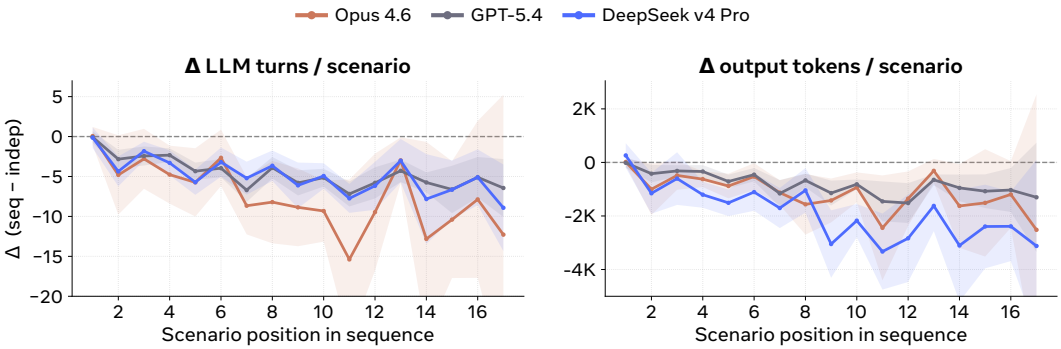


Figure 10: **GAIA2 effort by sequence position.** Change from independent to sequential (Δ = seq − indep) in LLM turns (left) and output tokens (right) per scenario; below zero means less effort sequentially. Bands show 95% confidence intervals.

Table 4: **Python share on GAIA2.** Fraction of tool calls that execute Python, with Search and Execution pooled.

	200K		32K	
Model	indep	seq	indep	seq
Opus 4.6	2.1%	8.9%	9.5%	19.1%
GPT-5.4	9.9%	15.3%	13.2%	24.2%
DeepSeek v4 Pro	2.6%	10.9%	6.3%	16.6%

(a) Fixed-bound helper script in `/home/agent/process_data.py`

```
import json, subprocess
def get_contacts(offset):
    r = subprocess.run(["contacts", "get-contacts", "--offset", str(offset)],
                       capture_output=True, text=True)
    return json.loads(r.stdout)

contacts = []
for off in range(0, 130, 10):
    contacts.extend(get_contacts(off)["contacts"])
# Assumes that 130 records are sufficient.
```

(b) Completeness-checked `search_contacts` helper

```
all_contacts, offset = [], 0
while True:
    data = get_contacts(offset)
    page = data["contacts"]
    all_contacts.extend(page)
    if offset + len(page) >= data["metadata"]["total"]:
        break
    offset += len(page)
# Re-implement the unsupported compound query
matches = [c for c in all_contacts
           if c["nationality"] == "Swedish" and c["age"] < 40 and not c["is_user"]]
```

Figure 11: **Persistent home-directory scripts on GAIA2 Search.** Condensed excerpts contrast fixed-bound pagination (a) with an Opus 4.6 agent's completeness-checked pagination and local implementation of a compound query unsupported by `search_contacts` (b). Both keep raw records out of the context window, while making correctness depend on the scripts' assumptions.

patches. For DeepSeek v4 Pro, the only frontier model with a clear accuracy loss, verification itself degrades: across matched runs, it modifies existing test files 542 times sequentially versus 56 times

independently, rewriting tests instead of repairing the code locally. In contrast, Muse Glimmer-30B nearly stops testing under persistence (Table 11), which Seq-RL largely prevents (Section 5.3).

Table 5: **SWE-Bench Pro effort and workflow at 200K.** Means per task, pooled over the 420 tasks of each run; parentheses give the relative change from independent to sequential. *To 1st edit*: turns before the first file edit. *Files*: distinct source files referenced. *Inspect/Edit/Test*: share of turns whose dominant action is inspection, editing, or testing.

		Effort		Exploration		Share of turns (%)		
Model	**Eval**	**Turns**	**Output tokens**	**To 1st edit**	**Files**	**Inspect**	**Edit**	**Test**
Opus 4.6	ind	49.3	14.6K	19.7	15.9	71.5	11.1	8.6
	seq	28.2 (−43%)	7.9K (−46%)	12.6	11.2	67.4	10.1	10.3
GPT-5.4	ind	24.1	31.3K	10.4	16.0	57.0	15.0	7.3
	seq	17.8 (−26%)	25.1K (−20%)	8.6	12.1	55.7	9.9	11.4
DeepSeek v4 Pro	ind	57.9	31.1K	22.0	14.8	72.5	10.8	3.5
	seq	28.3 (−51%)	12.6K (−60%)	13.0	10.1	64.8	6.0	7.3

B.3 Context pressure and window size

We test whether the window size explains the effects of persistence by shrinking it to 32K on GAIA2 and expanding it to 1M on SWE-Bench Pro. On GAIA2 at 200K, OpenClaw never summarizes the full history in either mode, while sequential runs compact old tool outputs an order of magnitude more often than independent runs (Table 6): the sequential effects arise without the context filling up. At 32K, independent runs also summarize and lose accuracy (Table 7), so the comparison is between two constrained settings, and the sequential gap remains model-dependent. Sequential runs still take fewer turns, but two of the three models now produce more output tokens (Figure 12), which suggests that they regenerate context lost to summarization.

Table 6: **Compaction on GAIA2.** Average compaction events per scenario, by model, split, context window, and evaluation mode.

		(a) Tool-output compactions				(b) Full-history summarizations			
		200K		32K		200K		32K	
Model	**Split**	indep	seq	indep	seq	indep	seq	indep	seq
Opus 4.6	Search	3.6	47.7	33.7	37.9	0	0	0.37	1.60
	Execution	1.3	22.0	14.7	53.6	0	0	0.55	1.83
GPT-5.4	Search	0.4	10.8	8.7	18.8	0	0	0.54	1.56
	Execution	0.3	25.2	12.4	35.3	0	0	0.51	1.58
DeepSeek v4 Pro	Search	2.2	38.7	19.0	25.2	0	0	0.44	1.81
	Execution	0.6	24.1	14.6	41.9	0	0	0.77	2.04

On SWE-Bench Pro, sequences are long enough that the 200K context is summarized repeatedly, whereas at 1M it retains nearly all history, growing to about 900K tokens (Figure 14). Independent runs never approach 200K, so they serve as the baseline for both windows. Keeping more history does not help: at 1M, avg@3, turns, and output tokens all fall further for every model (Figure 13, top), and the effort reduction grows toward the end of each repository sequence (Figure 13, bottom). Together with the 32K results, this suggests that the effects of persistence depend on how agents use accumulated context rather than on how much of it they can hold.

Table 7: **GAIA2 full metrics report at 32K** ($K = 5$)**.** Values are percentages; Δ = Seq. − Indep. For avg@5, ± is the half-width of the 95% cluster-bootstrap interval (Section A.3); other metrics are point estimates. Bold marks the best value per metric and split.

		GAIA2 Search			GAIA2 Execution		
Metric	**Model**	**Indep.**	**Seq.**	**Δ**	**Indep.**	**Seq.**	**Δ**
avg@5	Opus 4.6	82.8 ± 4.50	**88.4** ± 3.82	**+5.6** ± 3.22	**75.4** ± 6.95	**73.6** ± 4.87	−1.8 ± 4.63
	GPT-5.4	**87.9** ± 3.56	84.4 ± 4.15	−3.5 ± 4.86	67.6 ± 6.95	68.6 ± 7.18	**+1.0** ± 6.27
	DeepSeek v4 Pro	77.0 ± 4.49	75.3 ± 4.29	−1.7 ± 5.51	58.4 ± 7.10	56.7 ± 5.02	−1.7 ± 6.06
pass@5	Opus 4.6	95.0	**98.8**	**+3.8**	**90.5**	**92.6**	**+2.1**
	GPT-5.4	**97.5**	97.5	0.0	**90.5**	90.5	0.0
	DeepSeek v4 Pro	95.0	98.1	+3.1	89.5	85.3	−4.2
pass[5]	Opus 4.6	64.4	**71.9**	**+7.5**	**57.9**	**43.2**	−14.7
	GPT-5.4	**68.1**	58.1	−10.0	37.9	30.5	**−7.4**
	DeepSeek v4 Pro	42.5	40.0	−2.5	30.5	22.1	−8.4

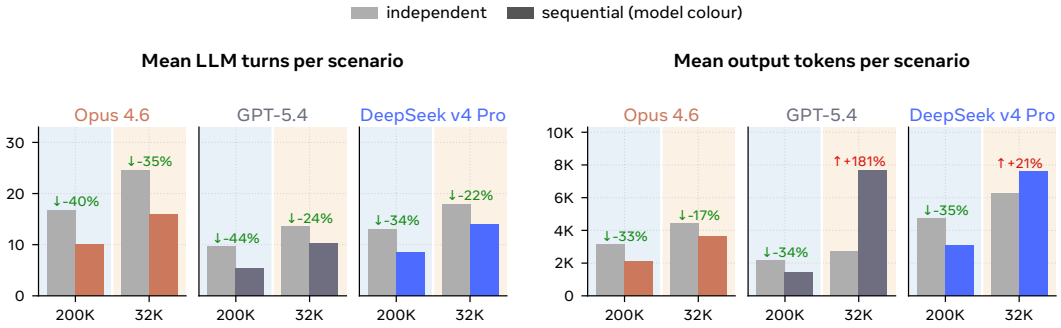


Figure 12: **GAIA2 effort at 200K and 32K.** Mean LLM turns (left) and output tokens (right) per scenario, with Search and Execution pooled; labels give the relative change from independent to sequential.

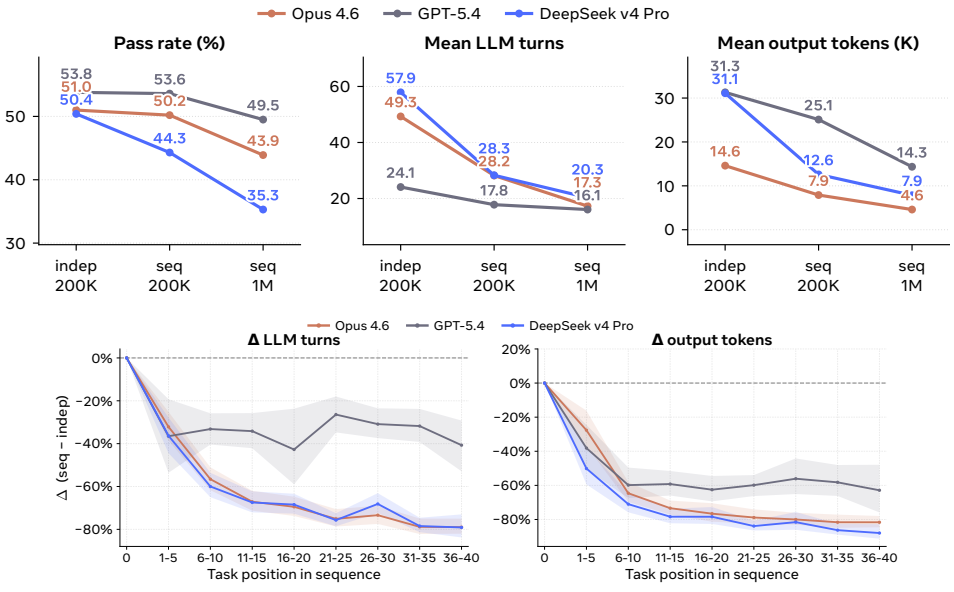


Figure 13: **SWE-Bench Pro at a 1M context window.** Top: avg@3, LLM turns, and output tokens for independent evaluation at 200K (equivalent to 1M, since independent runs never approach 200K) and sequential evaluation at 200K and 1M. Bottom: change from independent to sequential evaluation at 1M, by task position; bands show 95% confidence intervals.

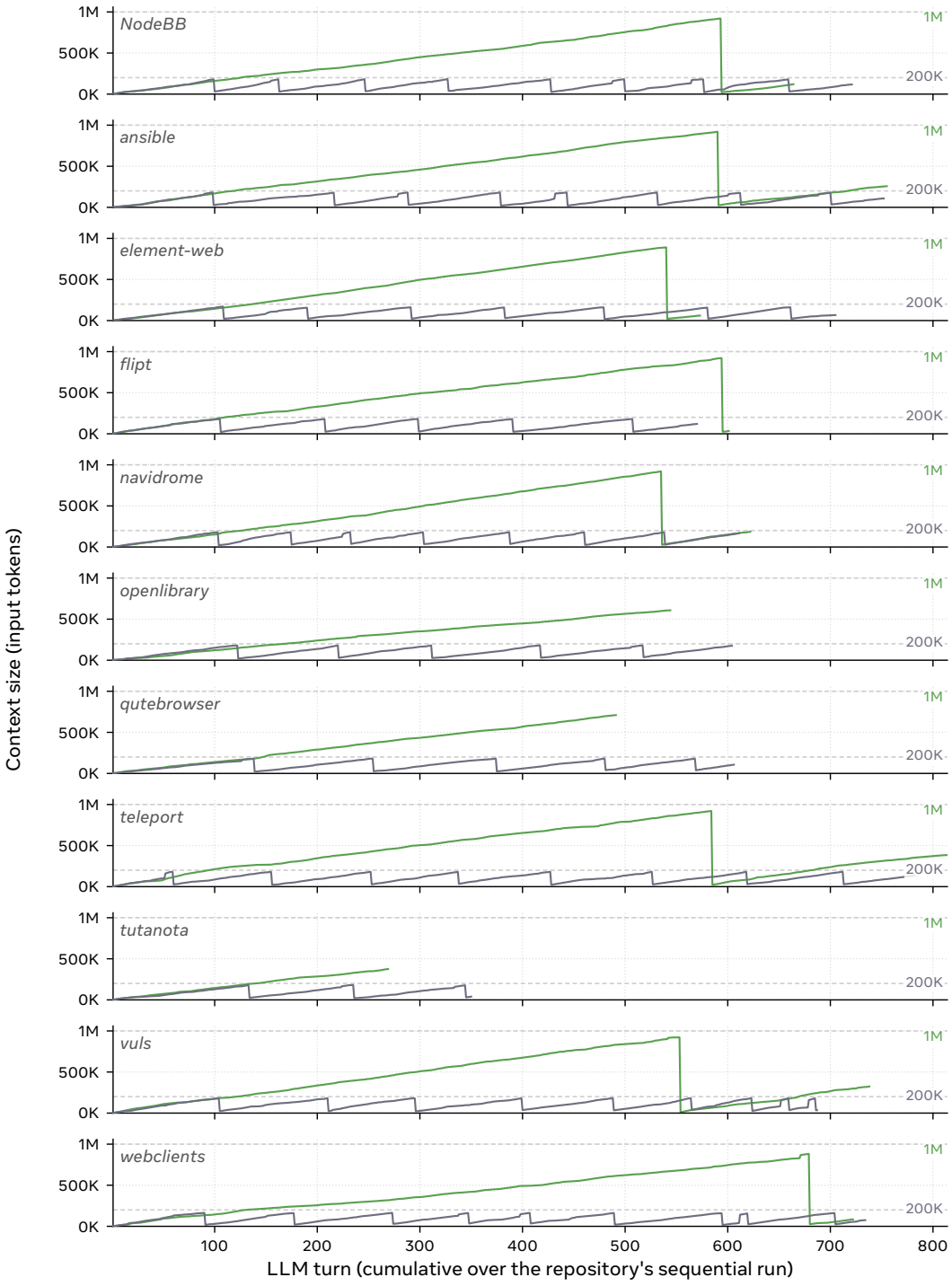


Figure 14: **SWE-Bench Pro context evolution.** Context size (input tokens) per LLM turn under sequential evaluation for GPT-5.4, one rollout per repository, with turns counted cumulatively over each repository's tasks. The 200K run is summarized frequently (median seven times per repository); the 1M run (green) grows to about 900K tokens and is summarized at most once.

C SEQ-RL: DETAILS AND ADDITIONAL RESULTS

This section gives the training data and settings (Section C.1), then reports training dynamics (Section C.2), the downstream-credit ablation (Section C.3), and complete results (Section C.4).

C.1 TRAINING DETAILS

Training data. The 27K training scenarios are Search- and Execution-style scenarios built in the GAIA2 format: they use the same universe structure, apps, and tools, and carry the same write-action DAG annotations and verifiers as the benchmark. They come from universes disjoint from the evaluation set and are chained into 1.6K compatible sequences of up to 50 scenarios with the procedure of Section A.1. Each sequence yields one training example with six trainable scenarios (Section 5.1).

Local reward. For continuation m at position i, let $y_{m,i} \in \{0, 1\}$ denote task success, and let M, O, and E be the numbers of matched oracle state-changing actions, oracle actions, and failure-causing extra actions. We define

$$p = \frac{M}{M+E}, \qquad q = \frac{M}{O}, \qquad F_2 = \frac{5pq}{4p+q}, \qquad r_{m,i} = (1-\alpha)y_{m,i} + \alpha F_2, \quad \alpha = 0.15. \tag{1}$$

We set $F_2 = 0$ when a denominator vanishes; when dense counts are unavailable, the local reward falls back to $y_{m,i}$.

Downstream credit. The training reward mixes the local reward with the mean local reward of the later scenarios in the same continuation:

$$R_{m,i} = \begin{cases} \beta r_{m,i} + (1-\beta)\dfrac{1}{N-i}\displaystyle\sum_{j=i+1}^{N} r_{m,j}, & i < N, \\ r_{m,N}, & i = N, \end{cases} \qquad \beta = 0.85. \tag{2}$$

Every trainable token produced for scenario i, including compaction tokens, receives $R_{m,i}$. Missing judgments receive zero reward; a rollout is discarded if more than half of its positions are unjudged, and truncated rollouts are not graded.

Baseline and advantage. Let $L_{m,i}$ be the number of trainable tokens for scenario i in continuation m. Seq-RL uses one baseline for all continuations and positions sharing a prefix:

$$b = \frac{\sum_{m=1}^{G}\sum_{i=1}^{N} L_{m,i}\, R_{m,i}}{\sum_{m=1}^{G}\sum_{i=1}^{N} L_{m,i}}, \qquad A_{m,i} = R_{m,i} - b. \tag{3}$$

This pooled, token-weighted baseline replaces a position-wise one. Token weighting, standard practice to keep response lengths from exploding, is the only normalization: advantages are not divided by a standard deviation, and we use no length penalty.

Configuration and compute. Each Seq-RL update contains $B = 32$ prefix groups with $G = 8$ continuations, or 256 continuation rollouts. Training updates the full model on 32 NVIDIA H200 nodes with a 128K-token context window, a learning rate of 2.6×10^{-6}, temperature 1.0, and top-$p = 1.0$; rollouts are generated asynchronously with a maximum policy lag of 16 trainer updates. Single-task RL uses the same recipe with no prefix and $N = 1$: each of the 9.6K trainable scenarios is sampled $G = 8$ times per epoch from a fresh session, for three epochs. All runs are data-matched (Section 5.1); because Seq-RL rollouts are longer, each Seq-RL run ($\beta = 0.85$ and $\beta = 1$) takes about 25K H200 GPU-hours, and single-task RL about 15K.

C.2 TRAINING DYNAMICS

Figure 15 shows the per-update mean local reward behind Section 5.2; the annotated values are means over the first and last ten updates. Single-task RL starts higher, since its scenarios begin in a fresh session, and stays near that level throughout training, whereas Seq-RL starts lower, after a 100K-token prefix, and its reward rises steadily over the three epochs.

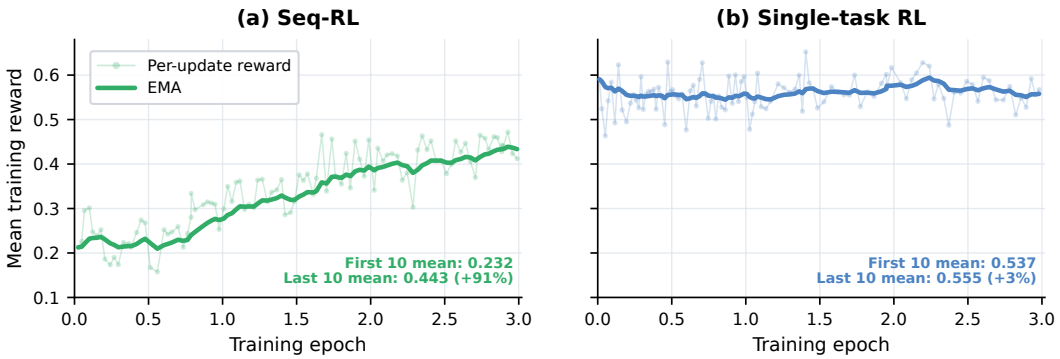


Figure 15: **Reward evolution under Seq-RL and single-task RL.** Light curves show the per-update mean local reward per scenario ($r_{m,i}$, Equation 1) and solid curves their exponential moving averages. Both methods start from the same checkpoint and train on the same scenarios, but Seq-RL scenarios follow a 100K-token prefix, whereas single-task RL scenarios start from a fresh session. We therefore compare how the reward evolves within each run rather than its absolute level.

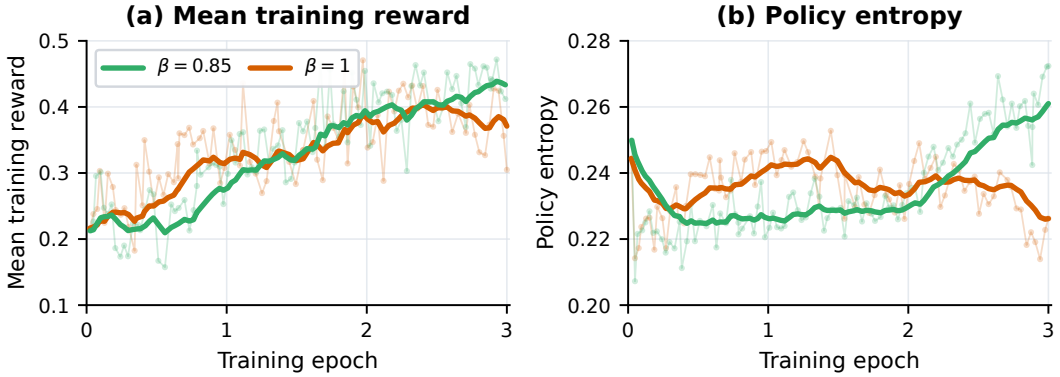


Figure 16: **Training dynamics with and without downstream credit.** We compare full Seq-RL ($\beta = 0.85$) with the local-reward-only objective ($\beta = 1$) in mean local reward per scenario (a), which is directly comparable between the two runs, and policy entropy (b). Light curves show per-update values and solid curves show exponential moving averages.

C.3 Downstream-credit ablation

To isolate downstream credit, we compare full Seq-RL ($\beta = 0.85$) with an otherwise identical local-reward-only variant ($\beta = 1$). With $\beta = 1$, the model trains on persistent histories but is rewarded only for the current scenario, so nothing directly rewards leaving the context or environment useful for later tasks. Its reward rises at first but plateaus earlier, and its policy entropy declines after about two epochs, whereas with $\beta = 0.85$ the reward keeps rising and entropy rises above that of $\beta = 1$ late in training (Figure 16). The difference is spread across all six trainable positions rather than driven by one (Figure 17). On held-out sequential GAIA2, full Seq-RL scores higher on every metric (Table 8), but with single-seed runs and overlapping avg@5 intervals, we read this as modest evidence and keep downstream credit in the final method.

C.4 Complete results

Table 9 reports all metrics for Muse Glimmer-30B (Baseline) and its RL and Seq-RL checkpoints. All results use a 200K window, whereas Seq-RL was trained at 128K, so its gains carry over to a longer context than seen in training. In the behavior tables, single-task RL produces the fewest output tokens and turns of the three checkpoints under persistence on GAIA2 (Table 10) and has the largest avg@K gaps, whereas Seq-RL stays close to the effort of Muse Glimmer-30B. On SWE-Bench Pro, Muse Glimmer-30B and single-task RL submit much smaller patches sequentially than independently,

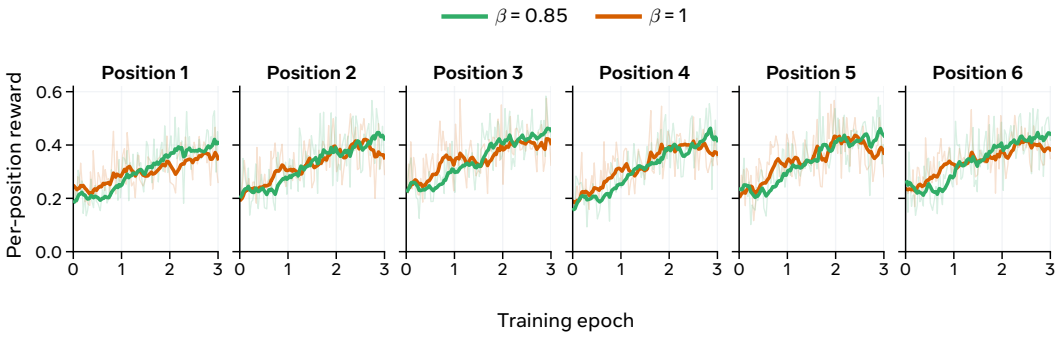


Figure 17: **Local reward by rollout position.** We compare full Seq-RL ($\beta = 0.85$) with $\beta = 1$, which removes the downstream-reward term, across the six trainable continuation scenarios. Light curves show the per-update mean local reward at each position and solid curves show exponential moving averages.

Table 8: **Effect of downstream credit on sequential GAIA2 performance.** Sequential evaluation at 200K ($K = 5$) after three epochs of training; values are percentages. For avg@5, $\pm$ is the half-width of the 95% cluster-bootstrap interval (Section A.3); pass^5 and pass@5 are point estimates. $\beta = 1$ removes the downstream-reward term.

	GAIA2 Search			GAIA2 Execution		
Objective	**pass^5**	**avg@5**	**pass@5**	**pass^5**	**avg@5**	**pass@5**
Full Seq-RL ($\beta = 0.85$)	78.8	92.5 $\pm$ 3.0	97.5	69.5	86.2 $\pm$ 6.1	95.8
No downstream reward ($\beta = 1$)	74.4	90.1 $\pm$ 3.0	96.9	66.3	84.7 $\pm$ 4.2	94.7

in line with their rise in empty patches (Figure 7c), while Seq-RL's patches shrink less relative to the gold patch (Table 11).

Table 9: **Full metrics report for Muse Glimmer checkpoints at 200K.** Baseline is Muse Glimmer-30B before RL. Values are percentages; Δ = Seq. $-$ Indep. For avg@K, $\pm$ is the half-width of the 95% cluster-bootstrap interval (Section A.3); other metrics are point estimates. GAIA2 uses $K = 5$ and SWE-Bench Pro $K = 3$.

		GAIA2 Search			GAIA2 Execution			SWE-Bench Pro		
Metric	**Method**	**Indep.**	**Seq.**	**Δ**	**Indep.**	**Seq.**	**Δ**	**Indep.**	**Seq.**	**Δ**
avg@K	Baseline	91.4$\pm$3.25	87.9$\pm$4.15	-3.5 ± 3.96	86.1$\pm$4.84	84.4$\pm$4.34	-1.7 ± 3.79	45.8$\pm$4.39	25.6$\pm$3.99	-20.2 ± 2.83
	RL	92.1$\pm$3.25	87.9$\pm$3.25	-4.3 ± 2.84	**88.6**$\pm$4.63	82.1$\pm$4.63	-6.5 ± 5.85	46.3$\pm$4.41	24.4$\pm$2.89	-21.9 ± 2.90
	Seq-RL	**92.8**$\pm$3.13	**92.5**$\pm$2.99	**-0.3 ± 2.27**	87.2$\pm$5.16	**86.2**$\pm$6.05	**-1.0 ± 7.01**	**48.0**$\pm$4.45	**42.0**$\pm$3.37	**-6.0 ± 3.36**
pass@K	Baseline	**97.5**	95.6	-1.9	**97.9**	97.9	0.0	53.8	37.6	-16.2
	RL	**97.5**	96.3	-1.3	**97.9**	**98.9**	**+1.0**	55.5	38.3	-17.2
	Seq-RL	96.9	**97.5**	**+0.6**	95.8	95.8	0.0	**56.2**	**53.1**	**-3.1**
pass^K	Baseline	79.4	66.9	-12.5	66.3	60.0	-6.3	36.0	14.0	-21.9
	RL	**82.5**	70.6	-11.9	**74.7**	57.9	-16.8	37.1	11.4	-25.7
	Seq-RL	81.9	**78.8**	**-3.1**	70.5	**69.5**	**-1.0**	**41.0**	**28.8**	**-12.1**

Table 10: **GAIA2 behavior of Muse Glimmer checkpoints.** Per-scenario output tokens, LLM turns, and Python use (the percentage of scenario runs that call Python at least once), for Search and Execution. Baseline is Muse Glimmer-30B before RL.

		Search			Execution		
Evaluation	**Checkpoint**	**Output tokens**	**Turns**	**Python use**	**Output tokens**	**Turns**	**Python use**
Independent	Baseline	8.46K	39.1	33.6%	5.99K	35.2	12.1%
	RL	5.25K	26.6	43.9%	4.80K	36.2	17.3%
	Seq-RL	6.70K	33.7	80.3%	5.41K	34.6	37.8%
Sequential	Baseline	3.96K	16.3	67.0%	4.39K	23.8	23.4%
	RL	2.56K	10.4	83.9%	3.38K	20.8	47.2%
	Seq-RL	4.11K	16.5	96.1%	4.20K	21.4	76.4%

Table 11: **SWE-Bench Pro behavior of Muse Glimmer checkpoints.** Per-task statistics under independent and sequential evaluation at 200K. Patch/gold is the number of edited lines (additions plus deletions) in the agent patch relative to the gold patch, in percent. Baseline is Muse Glimmer-30B before RL.

		Effort			Patch statistics		
Evaluation	**Checkpoint**	**Output tokens**	**# Turns**	**# Test turns**	**Added lines**	**Hunks**	**Patch/gold (%)**
Independent	Baseline	19.6K	94.0	3.8	97.8	8.5	81.8
	RL	14.8K	70.6	3.5	125.1	9.0	83.7
	Seq-RL	18.8K	89.0	4.4	134.5	9.1	91.3
Sequential	Baseline	3.8K	22.0	1.1	73.6	5.4	61.0
	RL	3.6K	20.0	1.2	59.6	5.7	54.6
	Seq-RL	8.7K	44.9	2.7	100.4	8.2	79.3