SCLATE: A SUBSTRATE FOR CONTINUAL-LEARNING AGENT TRAINING AND EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual-learning agents are systems of models, harnesses, and memory operating over long multi-session horizons. Evaluating and training them requires interleaving tasks with agent-side events such as session stop and start, crons, and memory consolidation. Yet existing benchmarks and training frameworks schedule only the benchmark's own events, leaving each benchmark and agent pair to build a custom scheduling loop. We present SCLATE, an execution substrate where benchmarks and unmodified agents each add their events to one open event scheduler through an adapter. A hybrid simulated clock runs these events on a shared timeline, flowing in real time while the agent works and skipping idle gaps, which compresses a month-long scenario into hours. SCLATE also serves as a rollout engine that runs any agent's harness and memory unmodified, recording the tokens and log probabilities of every model call through an in-container proxy. We port seven benchmarks to SCLATE and compare ten unmodified harness and memory configurations head to head on ten models. The comparison shows that an added memory system does not reliably beat the harness's native memory and that models differ widely in how they use the same harness and memory. We then post-train Qwen3.5-4B through unmodified harnesses and memory systems. The model learns to use both, reading 6.8× fewer file lines with a 16.7-point higher SWE-bench Verified pass rate, and writing richer memory records, while its held-out MetaClaw accuracy rises by up to 11.8 points.

1 INTRODUCTION

A continual-learning agent is a system comprising a memory layer that retains accumulated experience, a harness that manages tools and context, and model training that refines its behavior [29, 18, 3, 16, 26, 24, 12, 35]. Increasingly, agent platforms such as Claude Code, Codex, OpenClaw, and Hermes operate as such systems, learning across sequential sessions to accumulate a user's preferences, facts, habits, and workflow state. Across such a lifecycle, the agent and its memory system initialize within an environment, execute interactive tasks while an independent grading event scores state transitions, trigger background routines to consolidate memories across session boundaries, and resume across days or weeks on a progressing timeline. Evaluating and training these systems requires an execution substrate that runs an unmodified agent and its memory system alongside benchmark scenarios under reproducible conditions.

Yet no evaluation or training substrate can run an unmodified harness and memory across a multi-session continual-learning lifecycle. First, non-agentic benchmarks evaluate one-shot retrieval or prompt responses on pre-collected traces, lacking temporal horizons, session boundaries, or live agent loops [30, 15, 9, 13]. Second, closed-agent benchmarks model multi-session scenarios but bind execution to built-in assistants, testing memory or skill modules only within private runners rather than supporting external harnesses [19, 2, 36]. Third, agent-agnostic benchmarks accept external harnesses, yet keep event scheduling closed to the agent, dictating scenario turns while preventing agents from registering crons, session stop and start, or memory consolidation onto the shared timeline [25, 10, 31, 4]. On the training side, post-training rollout engines inherit these limitations by training agents only on existing single-session benchmarks, omitting session boundaries, persistent memory states, and asynchronous event simulation [32, 8, 33, 27]. Without a shared substrate to coordinate benchmark scenarios alongside agent-side events, researchers must build custom scheduling loops for every benchmark-agent pair, preventing controlled, head-to-head comparisons.

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

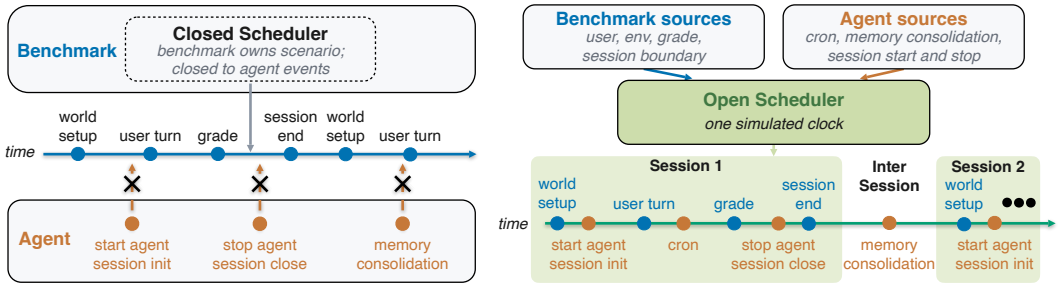


Figure 1: Continual-learning lifecycle across multi-session horizons. (a) Existing frameworks run only a built-in assistant or schedule only the benchmark's own events, leaving agent-side events off the timeline. (b) SCLATE's open event scheduler lets the benchmark and the agent register their events on one simulated clock, so the same runs serve multi-session evaluation and post-training.

To address these limitations, we introduce the Substrate for Continual Learning Agent Training and Evaluation (SCLATE), an open execution substrate where arbitrary continual-learning agents can be evaluated and trained across long-horizon lifecycles spanning days to weeks (Figure 1). SCLATE has three core mechanisms. First, in a continual-learning lifecycle both the benchmark and the agent produce events. SCLATE provides each with an adapter that registers those events on one open event scheduler, from scenario tasks to session stop and start, crons, and state save and restore. Second, to run multi-week scenarios in hours, the scheduler places these events on a hybrid simulated clock, visible to agent and memory processes, that flows in real time while the agent works and jumps across idle intervals between events. Third, to post-train models through unmodified harnesses and memory systems, an in-container proxy records every model call's tokens and log probabilities and tags each call with the harness or memory component that made it. SCLATE pairs this with a decoupled rollout engine that controls agent, memory, and container state across training iterations behind one training-framework-agnostic interface, which we demonstrate with Slime [37].

We validate SCLATE on seven benchmarks, MetaClaw [31], GAIA-2 [4], AppWorld [28], LongMemEval [30], PersonaMem [9], SWE-Gym [17], and SWE-bench Verified [10], spanning personal assistance, software engineering, and multi-session memory. With every harness and memory system running unmodified on the same benchmarks, SCLATE enables a head-to-head comparison of continual-learning agents across three harnesses, Claude Code, Hermes, and Codex, with their native memory or an external memory system, Mem0 [3], Hindsight [11], or GBrain [5], and ten models. The comparison shows that no harness or memory is best for every model and that an added memory system does not reliably beat the harness's native memory. Models also differ widely in how they use the same harness and memory, so continual-learning agents must be evaluated and trained with their model, harness, and memory together. SCLATE also supports post-training a model through its unmodified harness and memory. The model learns to use both, reading 6.8× fewer file lines while its SWE-bench Verified pass rate rises 16.7 points, and writing richer memory records, while its held-out accuracy on 30-workday MetaClaw rises by up to 11.8 points across configurations. SCLATE thus lets a continual-learning agent's model, harness, and memory be evaluated and trained together.

In summary, our contributions are the following.

- An open event scheduler (§3.1) that owns the timeline and runs events from both benchmarks and unmodified agents on it, each added through an adapter (§3.2), compressing month-long scenarios into hours while preserving event order and timing (§4.1).
- A rollout engine (§3.3) whose in-container capture proxy records every model call of an unmodified harness and memory, turning evaluation runs into training data.
- SCLATE, our implementation with seven ported benchmarks, used for a head-to-head comparison of ten harness and memory configurations on ten models (§4.2) and for post-training through unmodified harnesses and memory (§4.3).

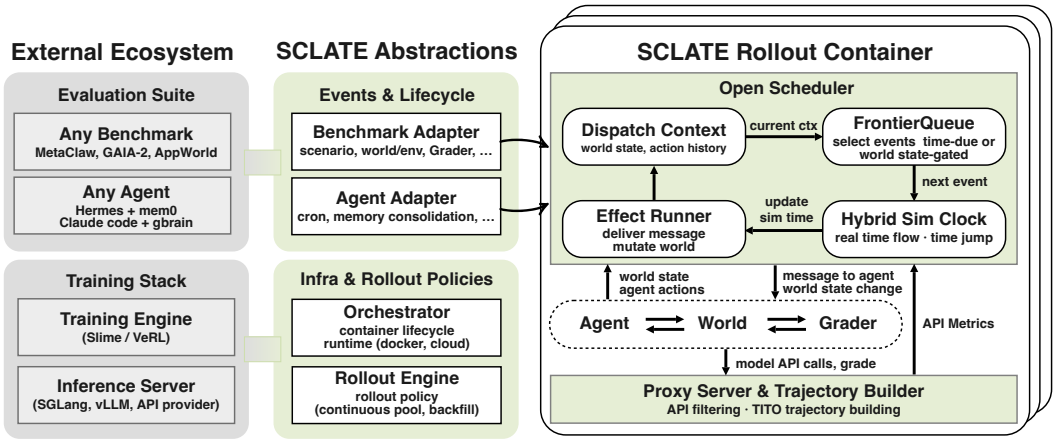


Figure 2: SCLATE architecture. Adapters decouple benchmarks and agent harnesses into a uniform event stream, while the host orchestration manages container lifecycles and training batches. Inside replicated sandboxes, an open event scheduler coordinates benchmark and agent events on a hybrid simulated clock, while an in-container proxy captures token-faithful trajectories for post-training.

2 Related Work

Continual-learning evaluation benchmarks. Evaluating continual-learning agents requires measuring how systems of models, harnesses, and memory persist and adapt across long multi-session horizons. Prior evaluation suites fall into three architectural tiers. First, non-agentic benchmarks score isolated dimensions such as memory retention and context compaction on pre-collected traces without an interactive agent loop [15, 30, 9, 13, 6], failing to evaluate the end-to-end behavior of continual-learning agents across session boundaries. Second, closed-agent benchmarks evaluate memory retention, skill acquisition, and model adaptation, but hardwire a built-in agent directly into the evaluation runner rather than supporting external harnesses and memory systems [19, 23, 2, 36]. Third, agent-agnostic benchmarks advance to evaluating unmodified agents [31, 4, 25]. However, they either lack an event scheduler entirely [31] or rely on benchmark-driven execution cycles that treat the agent as a passive responder [4, 25], leaving no mechanism to coordinate agent and memory events across continual-learning lifecycles. SCLATE decouples the event scheduling control loop from the benchmark into an open scheduler, supporting the broader continual-learning lifecycle events of agents and benchmarks.

Rollout engines for agent post-training. A growing line of systems scales agent post-training by decoupling rollout execution from distributed training loops [8, 27, 32, 37, 1, 20, 34]. To train unmodified agents, recent frameworks intercept model calls via network proxies [32, 8, 33] or wrap execution loops in standardized abstractions [27]. However, training continual-learning agents requires carrying session state across container restarts and attributing each model call to the harness or memory component that made it. SCLATE addresses these requirements by providing abstractions to manage rollout state across container lifecycles and the flexibility to route and attribute model calls by component, natively supporting continual-learning agents at scale while remaining agnostic to benchmarks and training frameworks. Table 3 in Appendix A compares SCLATE with these systems.

3 The SCLATE Substrate

Architecture overview. SCLATE is an execution substrate for continual-learning agent evaluation and post-training across multi-session horizons (Figure 2). SCLATE is organized into three decoupled tiers, an external tier of benchmarks and agent harnesses, a host orchestration plane that manages container lifecycles and training batches, and replicated container sandboxes that execute scenarios in isolation. The system provides standardized abstractions for event scheduling, agent lifecycles, and sandboxed execution, allowing new benchmarks and external agent harnesses to plug in without modifying substrate code. At its core, an open event scheduler coordinates scenario progression, agent-side routines, and task grading on a shared hybrid simulated clock. An in-container

capture proxy, through which all model calls pass, pairs live evaluation with on-policy trajectory capture. Throughout the paper, an unmodified harness is run without changes to its source code and integrates only through an adapter, configuration files such as hook settings, and environment variables such as the model endpoint pointed at the capture proxy and the libfaketime preload.

3.1 Event Scheduling Loop and Hybrid Clock

A continual-learning lifecycle interleaves benchmark events (scenario tasks, environment mutations, session boundaries) with agent-side routines (process bootstrap, background crons, memory consolidation, and state handoffs). Running it in wall-clock time is prohibitively slow, while naive discrete-event simulation breaks execution by advancing time while an agent is still executing tools. SCLATE avoids both with an open event scheduler to which benchmark adapters and agent adapters register event sources, coordinated over a hybrid simulated clock.

Event scheduling loop. At each tick, the scheduler snapshots simulated time, world state, agent actions, and grading outcomes into a DispatchContext. The FrontierQueue then selects the registered event sources that are due, by time or by a condition on the world state, agent actions, or grading outcomes. The simulated time controller advances the clock, and the Effect Runner applies the chosen event, delivering a prompt or notification, mutating the world, or starting, stopping, saving, or restoring the agent and its memory. Appendix B gives the full loop (Algorithm 1) and the catalogs of event sources and effect primitives (Tables 5 and 6).

Hybrid simulated clock. The clock flows with real time while the agent is busy or a background job holds the clock, so that tools, test suites, and internal timers run naturally. Once nothing is busy, the future event stream is fixed, and the clock jumps across the idle interval to the earliest scheduled event, compressing overnight gaps and session boundaries into pure execution time, as traced on one MetaClaw workday in Figure 9 (Appendix C.4) and measured in Table 7. libfaketime [14] makes standard system-time calls return the simulated time for every dynamically linked harness and memory process evaluated here, so native crons in Claude Code and Hermes follow the scenario timeline without source changes (Figure 8 in Appendix C.2).

3.2 Pluggable Benchmark and Agent Adapters

SCLATE coordinates the benchmark and agent lifecycles through two adapters, the Benchmark Adapter and the Agent Adapter, both of which register event sources with the scheduler (Table 4 in Appendix B). The adapters translate scenario progression and agent routines into the shared event language, enabling controlled, head-to-head evaluation of unmodified harnesses across benchmarks.

Benchmark adapter and scenario composition. A benchmark needs its world state set up, benchmark events delivered, and turns graded. The Benchmark Adapter plugs a benchmark into SCLATE through four components (Table 4). Its setup routine establishes the baseline world state at container start, configuring code repositories, seeding databases, and starting benchmark application servers. Its source manifest turns the benchmark events (scenario tasks, environment mutations, and session boundaries) and the graded turns into SCLATE's event sources. An optional world provider exposes a read-only view of application state to event sources and graders through the DispatchContext. A grading policy declares which turns are graded and when, and an independent grader scores them against the task rubric, so the scheduler carries no benchmark-specific grading logic.

Agent adapter and state persistence. Continual-learning agents evolve across multi-session lifecycles where memory systems persist durable state, while the agent process stops and starts at session boundaries. The Agent Adapter connects the open scheduler to an unmodified harness and its memory system through interfaces for the session lifecycle, message delivery, and state save and restore, and registers the agent's own event sources with the scheduler (Table 4). During container startup and session transitions, the substrate polls the adapter's readiness probe until the memory system and the agent process can receive turns, the adapter delivers each message the scheduler dispatches into the live agent binary without blocking and parses the agent's response from it, and the adapter ends each session at its boundary, where hooks attached to session-start and session-end events can flush session state, persist memory, or run custom scripts. The adapter also registers the agent's own event sources, such as session-start and handoff prompts, background crons, and memory consolidation, while crons and wakeups that a harness schedules internally run on the simulated clock and

keep it flowing. The scheduler performs state persistence across sessions by consulting the adapter's memory specification, which declares the agent and the memory-system state separately and defines how to flush, snapshot, and restore agent files and sidecar databases.

3.3 Host Orchestration and Rollout Infrastructure

Scaling multi-session agent evaluation and post-training rollouts requires coordinating parallel execution across heterogeneous infrastructure while isolating complex benchmark environments. Standardized container environments initialize reproducible workspaces across parallel instances while preventing evaluation leakage. The host control plane organizes execution through the Orchestrator, which manages scenario configurations and container lifecycles, and the Rollout Engine, which coordinates distributed rollout collection for post-training. This division of responsibilities keeps benchmark dependencies, agent execution, and trajectory capture self-contained across parallel runs.

Pluggable runtime and rollout abstractions. The Orchestrator manages container lifecycles through a unified Runtime protocol that presents the same execution semantics across local and distributed infrastructure. This protocol defines typed interfaces for container provisioning, job submission, non-blocking health polling, and artifact retrieval, with adapters supporting local Docker engines or a managed cloud cluster. Building on this runtime foundation, the Rollout Engine provides abstractions to manage stateful container lifecycles across multi-session horizons. Leveraging the adapter's memory specification, the engine exposes the adapter's save and restore primitives, selected by scope, to move agent and memory state across container instances, while leaving task selection and training steps to the external trainer stack. This standardized interface abstracts parallel stateful execution into structured trajectory batches, interfacing with external post-training frameworks, such as reinforcement learning and on-policy distillation, without coupling lifecycle management to specific training algorithms. SCLATE validates this complete host abstraction across local Docker engines and a managed cloud cluster for runtime execution, coupling the Rollout Engine with Slime [37] as an example distributed training stack.

In-container capture proxy and trajectory assembly. To support post-training without modifying harness code, SCLATE deploys a capture proxy server directly inside each rollout container. The capture proxy operates as the dedicated model API gateway inside the container, recording prompt tokens, completions, and log probabilities (example in Figure 14) while routing requests to local inference engines or cloud providers based on model type and API keys. Through configurable tagging rules over API keys, system prompt signatures, and tool schemas, the proxy categorizes each invocation, allowing downstream trainer adapters to distinguish the agent's own turns from auxiliary calls, such as the web-search calls excluded from SWE-Gym training. The proxy assembles the recorded calls into token-level trajectories, and on the host the Rollout Engine launches rollouts and gathers these trajectories with their log probabilities and grading verdicts into training batches for external post-training frameworks.

In-container privilege separation. SCLATE keeps evaluation data out of the agent's reach by running the agent under a separate unprivileged user, so the agent cannot read ground truth or grading hints. The daemon user runs the event scheduler and benchmark services and owns the scenario ground truth, run artifacts, and grading and scheduling code, in directories the agent user cannot list or read. The agent user runs the harness and its shell tools in its own workspace and reaches the daemon only through message channel files that agent and daemon users can read and write.

4 Experiments

We now evaluate whether SCLATE achieves our design goals.

- Can SCLATE's open scheduler run asynchronous benchmark scenarios and agent-side events on one simulated clock while compressing idle time and preserving event order (Section 4.1)?
- Can SCLATE compare harness and memory configurations head to head across models, and what does the comparison reveal about how models use their harness and memory (Section 4.2)?
- Can SCLATE post-train models through unmodified harnesses and memory systems using the same runs it evaluates, including across multi-session lifecycles (Section 4.3)?

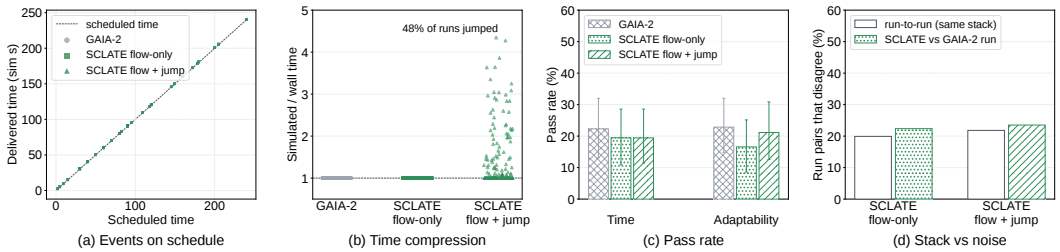


Figure 3: Parity between the upstream Gaia2-CLI stack and SCLATE, with the same agent and judge. (a) Every scheduled environment event is delivered on time on all three stacks. (b) On the Time scenarios, idle-time jumps compress simulated time by up to 4.3×; the label gives the share of runs that clock jumped. (c) Pass rate on 25 Time and 25 Adaptability scenarios with Haiku-4.5, each scenario repeated 7 times on every stack (175 runs per bar, 1,050 runs in total). (d) An SCLATE run and a GAIA-2 run of the same scenario disagree about as often as two runs of the same stack.

Table 1: MetaClaw pass accuracy over 30-workday arc (346 rounds) across ten models and ten harness and memory configurations. Bold marks the best accuracy within each harness-model pair.

	Claude Code					Hermes				Codex
Model	No-Mem	Native	Mem0	Hindsight	GBrain	No-Mem	Native	Mem0	Hindsight	Native
Haiku-4.5	40.2%	46.0%	**69.7%**	58.7%	56.7%	43.6%	59.8%	**61.9%**	61.0%	44.8%
Sonnet-5	43.9%	51.2%	57.5%	50.9%	**62.1%**	43.1%	47.4%	**59.0%**	50.3%	56.7%
Opus-4.6	48.8%	65.0%	66.5%	71.1%	**74.0%**	46.5%	**68.5%**	65.0%	66.8%	51.5%
Opus-4.8	48.6%	60.4%	66.8%	**74.0%**	67.6%	45.4%	64.2%	**64.7%**	61.9%	48.6%
Gemini-3.7F	63.9%	79.2%	79.2%	**81.8%**	79.5%	73.4%	77.8%	79.5%	**80.1%**	73.4%
GPT-5.4	40.5%	50.3%	**56.4%**	53.2%	53.8%	41.0%	53.8%	**68.8%**	57.8%	40.8%
GLM-5.3F	46.8%	65.0%	**66.8%**	65.9%	66.2%	47.1%	60.4%	**67.3%**	64.5%	60.4%
Qwen3.8-FN	49.7%	69.7%	69.4%	66.8%	**74.3%**	52.9%	**71.4%**	63.3%	68.2%	70.8%
DS-Vision	45.4%	54.6%	**66.8%**	63.0%	59.0%	50.0%	**69.1%**	63.0%	62.1%	53.5%
DS-V4.1	52.0%	64.2%	74.3%	74.3%	**74.9%**	59.3%	74.0%	**74.6%**	68.8%	73.7%

Experimental setup. We evaluate SCLATE across seven ported public benchmarks spanning personal assistance, software engineering, and multi-session memory. Detailed workload porting, dataset splits, and environment specifications are in Appendix D.1.

Models. We use ten proprietary and open models: Haiku-4.5, Sonnet-5, Opus-4.6, Opus-4.8, Gemini-3.7F, GPT-5.4, GLM-5.3F, Qwen3.8-Flash-Next (Qwen3.8-FN in tables), DeepSeek-Flash-Vision, and DeepSeek-V4.1-Flash. Models are queried through API provider endpoints or self-hosted serving endpoints, using the same reasoning effort for a model under every harness (Table 8 in Appendix D.1).

Harnesses and memory systems. We evaluate three agent harnesses (Claude Code, Hermes, and Codex) paired with three memory backends (Mem0 [3], Hindsight [11], and GBrain [5]). Native refers to each harness's built-in memory (Claude Code auto-memory and Hermes skills with notes), and No-Mem turns it off while keeping the harness tools, including session search over prior transcripts. Mem0, Hindsight, and GBrain run on top of native memory and make their own extraction and consolidation calls with the same model as the agent. Every harness introduces session stop and start events. Mem0 and Hindsight retain and recall memories during task execution, while GBrain registers memory consolidation events between sessions.

Workloads. Our ported benchmark suite includes MetaClaw [31] (30-workdays spanning 40 calendar days, 346 interactive rounds), GAIA-2 [4], AppWorld [28], two converted recall suites (LongMemEval [30], PersonaMem [9]), SWE-Gym [17], and SWE-bench Verified [10].

4.1 Validating the Open Event Scheduler

We validate SCLATE's open event scheduler against upstream GAIA-2 by porting its scenarios as an ordinary benchmark adapter. GAIA-2 schedules its events asynchronously, so they can arrive while the agent is working, and triggers them by time or by world state, such as a reply that follows the agent's own message after a set delay [4]. Its scheduler is closed, owning the clock and advancing

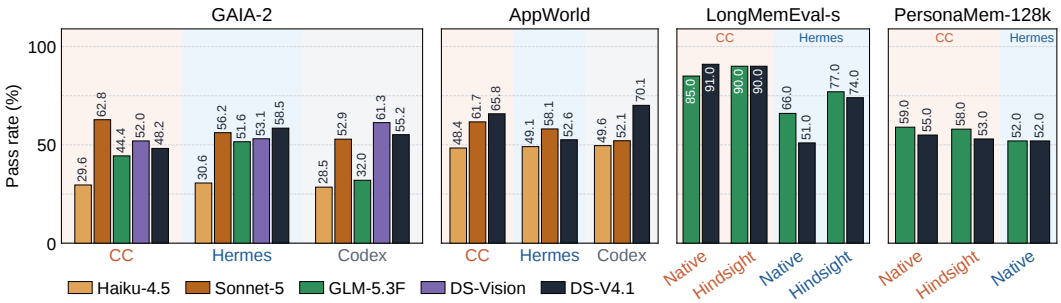


Figure 4: Pass rate (%) of each model under each harness and memory configuration; CC is Claude Code. **Left: GAIA-2 and AppWorld,** with each harness on its native memory. No harness is best for every model. **Right: LongMemEval-s and PersonaMem-128k,** with Claude Code and Hermes on native memory or an added memory system. Claude Code recalls more than Hermes on LongMemEval-s, and the two are close on PersonaMem-128k. Hermes with Hindsight is not shown on PersonaMem-128k, where its per-turn memory updates exceed the 900 s turn limit.

it in real time or accelerated only when the agent calls a wait tool, so only the scenario can schedule events. We run the same Hermes agent with Haiku-4.5 and the same judge on 25 Time and 25 Adaptability scenarios, on upstream GAIA-2 and on SCLATE with its clock either only flowing or flowing and jumping (Figure 3). Every scheduled environment event is delivered on all three stacks, in the same order and within 0.9 s of its scheduled time at the median and 2.0 s at most (Figure 3a). The three stacks reach overlapping pass rates (Figure 3c, with the grading in Appendix C.3), and an SCLATE run and a GAIA-2 run of the same scenario disagree about as often as two runs of the same stack (Figure 3d), so the port changes outcomes no more than rerunning the agent does. Across all evaluation, the scheduler fires all 13,050 GAIA-2 environment events, and 98.7% of the checkable ones at the delay the scenario specifies (Table 7, with walkthroughs in Figures 10–12).

We then measure how far the hybrid clock compresses scenarios and check whether the harness and memory run on simulated time. On GAIA-2 Time scenarios the clock jumps in about half the runs, compressing a run by up to $4.3\times$ (Figure 3b), and across all 100 MetaClaw runs of Table 1 a 30-workday arc spanning 40 calendar days finishes in a median of 5.9 real hours, a median speed-up of $162\times$ ($142\times$ over all runs combined) (Table 7 in Appendix C.1). Under Claude Code, Codex, and Hermes, the simulated date reaches the harness's system prompt, its shell clock queries, and the timestamps that Mem0 and Hindsight record (Figure 8). Appendix C.4 traces the mechanism on individual GAIA-2 and MetaClaw runs.

4.2 Evaluating Continual-Learning Agents

Continual-learning evaluation on MetaClaw. Upstream MetaClaw [31] runs a fixed memory system and simulates time through dates injected into prompts. SCLATE's MetaClaw adapter replaces those dates with the simulated clock, so unmodified harnesses (Claude Code, Hermes, Codex) and memory systems (Mem0, Hindsight, GBrain) run together across 30 simulated workdays.

Table 1 reports the full grid of ten harness and memory configurations across ten models over 346 rounds. Every memory configuration outperforms the no-memory baseline of its harness for all models, but no single memory system is best for every model under either harness. Against native memory alone, an added memory system scores lower in 11 of 50 cells, 8 of them under Hermes, whose native skills and notes are strong. On Haiku-4.5, harness workflows shape task success, lifting Hermes Native to 59.8% versus 46.0% for Claude Code, which trajectory inspection attributes to dedicated skill-management tools and a hook that requires verification after tool calls. On most other models this harness gap narrows or reverses, with Claude Code Native matching Hermes Native within 1.4 percentage points (pp) on Gemini-3.7F and leading it on Sonnet-5, though DS-Vision shows a similar 14.5 pp gap. Gemini-3.7F also uses the harness's session-search tool to inspect prior session transcripts, spending 7.6% of its tool calls (2,254 of 29,550) on session search versus at most 1.6% for other models, and is the only model to search in all ten configurations (Figure 13 in Appendix D.2). This behavior recovers procedural rules from earlier sessions without any memory system, consistent with its lead in no-memory accuracy, while the other models rarely call the same tool. These results show that an added memory system does not reliably beat the harness's native memory and that models differ widely in how they use the same harness and memory.

	In-distribution (days 1–20)				Held-out (days 21–30)			
Harness & Memory	Base	Trained	Δ (pp)	p	Base	Trained	Δ (pp)	p
CC & no-memory	37.4%	52.4%	+15.0	<0.001	9.2%	16.8%	+7.6	0.039
CC & Mem0	42.7%	67.4%	**+24.7**	<0.001	10.1%	21.8%	**+11.8**	0.023
CC & Hindsight	51.1%	52.0%	+0.9	0.91	16.0%	16.8%	+0.8	0.83
Hermes & no-memory	29.1%	36.6%	+7.5	0.27	8.4%	18.5%	**+10.1**	0.047
Hermes & Mem0	31.3%	58.6%	**+27.3**	0.004	8.4%	16.0%	+7.6	0.17
Hermes & Hindsight	39.6%	58.6%	+18.9	<0.001	10.1%	20.2%	**+10.1**	0.051

Table 2: MetaClaw post-training on a Qwen3.5-4B model across six configurations. Columns show base and post-trained (epoch 5) accuracy and the improvement (Δ) on the in-distribution and held-out partitions. p is a paired wild cluster bootstrap-t test with simulated days as clusters.

GAIA-2 and AppWorld results. On the 800 GAIA-2 scenarios (Figure 4, per category in Table 11), Search and Execution are the easiest categories in all 15 harness–model cells, as in the original release, while Adaptability and Time, whose scenarios depend on events delivered during the run, stay low in every cell (at most 37.5% and 34.4%). The strongest model differs by harness, with Sonnet-5 leading under Claude Code (62.8%), DS-V4.1 under Hermes (58.5%), and DS-Vision under Codex (61.3%), and because generation consumes simulated time as in GAIA-2's default time mode, these scores also reflect each harness's and endpoint's latency. On AppWorld, harnesses reach 48.4%–49.6% on Haiku-4.5, 52.1%–61.7% on Sonnet-5, and up to 70.1% on DS-V4.1 (Figure 4).

Multi-session recall benchmarks. We convert two memory suites, LongMemEval and PersonaMem, into multi-session benchmarks. Their upstream formulations score retrieval over pre-collected conversation logs in a single static prompt, with no process boundaries, background maintenance routines, or live tool use. Our adapters split each conversation history into sessions on consecutive simulated days, stop the agent process between sessions, and ask the question in the last session, so the agent must carry earlier facts through its native memory or a memory system, while the dataset content, questions, and ground-truth answers stay unchanged. On LongMemEvals, Claude Code's native memory outscores Hermes's, 85 and 91 against 66 and 51 on GLM-5.3F and DS-V4.1 (Figure 4). Hermes's built-in memory has a small size cap and rejects new facts once full, so most facts are never stored, and adding Hindsight, which has no such cap, lifts Hermes's accuracy to 77 and 74 on GLM-5.3F and DS-V4.1.

4.3 Post-Training Continual-Learning Agents

Multi-session post-training on MetaClaw. We next test whether post-training through the unmodified harness and memory teaches the model to use them. We post-train a Qwen3.5-4B model [21] on the 30-workday MetaClaw by on-policy distillation (OPD) under Claude Code and Hermes, each with no memory, Mem0, and Hindsight. The student generates every rollout through the unmodified harness and is trained on the per-token log-probability gap to a Qwen3.5-35B-A3B teacher that is given the round's corrective feedback, which the student never sees (Appendix E.2). Under Hindsight, fact-extraction calls also run on the student and pass through the capture proxy, so they are trained alongside the agent's own turns and post-training shapes both how the model acts and what it writes to memory. We train on simulated days 1–20 and evaluate base and trained checkpoints on all 346 rounds (Table 2). On the held-out days 21–30, accuracy rises in all six configurations, by 7.6 to 11.8 pp in five of them, and the gain is significant ($p < 0.05$) in three. We report the change from base to the final epoch 5, and Figure 15 in Appendix E.2 shows held-out accuracy at every epoch.

We also train Hermes with Mem0 on the first two-thirds of each day's rounds and evaluate on the last third, to test whether post-training improves unseen rounds without the shift to new, harder days that the day split introduces. On these 104 rounds, accuracy rises by 20.2 pp ($p = 0.007$, Appendix E.3).

Training changes how the model uses its memory system. We compare the memory writes of the base and epoch-5 trained checkpoints on Hermes with Hindsight to see what the model does differently, beyond what accuracy shows (Appendix F). After every agent turn, Hindsight uses the same model as the agent to extract facts from the Hermes transcript into a fixed schema (statement, date, `Involving` entities, optional purpose clause), so the facts reflect the model being trained. The trained model writes longer facts, and nearly all of the added length is metadata, which doubles

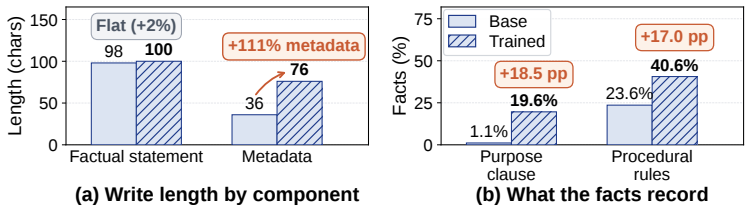


Figure 5: Memory facts written by the base and trained models as Hindsight's fact extractor on Hermes (5,027 and 3,617 facts). (a) Median length: training lengthens the metadata (+111%) but not the statement (+2%). (b) Facts containing the optional purpose clause or a procedural rule.

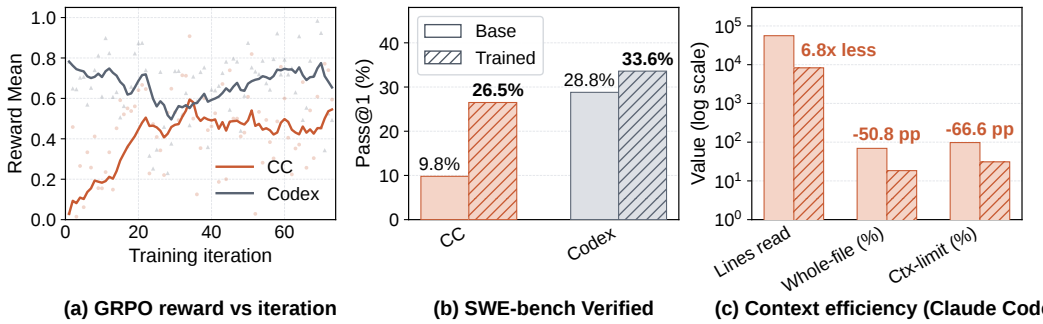


Figure 6: SWE-Gym post-training and downstream SWE-bench Verified generalization on SCLATE. (a) GRPO rollout mean reward across 73 training iterations with exponential moving averages (α=0.15). (b) Pass@1 on SWE-bench Verified comparing base and post-trained checkpoints across harnesses. (c) Context use under Claude Code on 42 matched tasks, where the trained model reads fewer lines, avoids whole-file reads, and rarely hits the context limit.

from 36 to 76 characters while statements stay near 100 (Figure 5a). It also fills the purpose clause, stating why a fact was worth recording, in 19.6% of facts versus 1.1% before (Figure 5b). Facts naming any of the benchmark's five procedural rules rise from 23.6% to 40.6% (Figure 5b), a gap that holds at every fact length (Figure 18a in Appendix F). Facts whose `Involving` field names the user fall from 65.3% to 31.5% while those naming the agent rise from 15.7% to 47.0%, so the trained model records what the agent did where the base model records what it was told. Appendix F gives length-matched controls, per-convention breakdowns, and verbatim examples.

Single-session post-training on SWE-Gym. We also validate SCLATE's post-training by executing GRPO [22] on SWE-Gym paired with the same Qwen3.5-4B model across unmodified Claude Code and Codex harnesses, where the in-container capture proxy records the model's tokens and the grader's verdicts become GRPO rewards. Over 73 iterations, the Claude Code rollout reward rises from 21.4% to 53.2% (first versus last ten iterations), while Codex stays near 69% (Figure 6a). The trained models transfer to SWE-bench Verified, lifting Pass@1 from 9.8% to 26.5% under Claude Code and from 28.8% to 33.6% under Codex (Figure 6b). On 42 matched evaluation tasks, the trained Claude Code model reads 6.8× fewer lines, cuts whole-file reads from 69.2% to 18.4%, and hits the context limit on 31.0% of tasks instead of 97.6% (Figure 6c).

5 Conclusion

We presented SCLATE, an execution substrate built on an open event scheduler that lets benchmarks and agents place their events on one simulated timeline, so each needs only its own adapter instead of a custom scheduling loop for every pair. The same scheduler runs month-long MetaClaw and asynchronous GAIA-2 scenarios without benchmark-specific scheduling logic, and its hybrid clock compresses 40 calendar days into a median of 5.9 hours with events kept on schedule and in order. Comparing ten harness and memory configurations across ten models shows that no harness or memory is best for every model and that models differ widely in how they use the same harness and memory, so continual-learning agents must be evaluated with all three together. Finally, SCLATE enables post-training a model through its unmodified harness and memory, and the trained model adapts to both, working within its harness's context limit and writing richer memory records.

AI USE STATEMENT

Generative AI tools were used to assist with coding and edit the paper for grammar, clarity, and concision. All experimental implementations, systems designs, mathematical formulations, and textual claims were verified and validated by the authors, who take full responsibility for the contents of this paper.

REFERENCES

[1] Shiyi Cao, Dacheng Li, Fangzhou Zhao, Shuo Yuan, Sumanth R. Hegde, Connor Chen, Charlie Ruan, Tyler Griggs, Shu Liu, Eric Tang, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Skyrl-agent: Efficient rl training for multi-turn llm agent, 2025. URL https://arxiv.org/abs/2511.16108.

[2] Jiayang Cheng, Dongyu Ru, Lin Qiu, Yiyang Li, Xuezhi Cao, Yangqiu Song, and Xunliang Cai. Amemgym: Interactive memory benchmarking for assistants in long-horizon conversations. *arXiv preprint arXiv:2603.01966*, 2026. ICLR 2026.

[3] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

[4] 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.

[5] Gbrain Team. Gbrain: Garry's opinionated OpenClaw/Hermes agent brain. https://github.com/garrytan/gbrain, 2026. GitHub repository, commit 872c3d6a.

[6] Xixuan Hao, Zeyu Zhang, Zehao Lin, Yihang Sun, Ziliang Guo, Xichong Zhang, Yuxuan Liang, Feiyu Xiong, and Zhiyu Li. Memops: Benchmarking lifecycle memory operations in long-horizon conversations, 2026. URL https://arxiv.org/abs/2607.12893.

[7] Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments. GitHub repository, https://github.com/harbor-framework/harbor, 2026.

[8] Zhiyuan He, Siwei Zhang, Zhiwen Zhou, Yuqing Yang, Yu Kang, Yuge Zhang, Luna K Qiu, Tin Yan Tsui, Jiahang Xu, and Chong Luo. Agent lightning v1. 0: Towards harnessed agentic rl. *arXiv preprint arXiv:2608.17528*, 2026.

[9] Bowen Jiang, Zhuoqun Hao, Young-Min Cho, Bryan Li, Yuan Yuan, Sihao Chen, Lyle Ungar, Camillo J Taylor, and Dan Roth. Know me, respond to me: Benchmarking llms for dynamic user profiling and personalized responses at scale. *arXiv preprint arXiv:2504.14225*, 2025.

[10] 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.

[11] Chris Latimer, Nicoló Boschi, Andrew Neeser, Chris Bartholomew, Gaurav Srivastava, Xuan Wang, and Naren Ramakrishnan. Hindsight is 20/20: Building agent memory that retains, recalls, and reflects, 2025. URL https://arxiv.org/abs/2512.12818.

[12] Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

[13] Shuyue Stella Li, Bhargavi Paranjape, Kerem Oktar, Zhongyao Ma, Gelin Zhou, Lin Guan, Na Zhang, Sem Park, Lin Chen, Diyi Yang, et al. Horizonbench: Long-horizon personalization with evolving preferences. *arXiv preprint arXiv:2604.17283*, 2026.

[14] libfaketime Team. libfaketime modifies the system time for a single application. `https://github.com/wolfcw/libfaketime`, 2026. GitHub repository, version v0.9.12.

[15] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2402.17753.

[16] Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

[17] Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. arXiv:2412.21139, 2024.

[18] Joon Sung Park, Joseph O'Brien, Carrie Jun 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*, pp. 1–22, 2023.

[19] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[20] Prime Intellect. Prime-rl: Async rl training at scale. GitHub repository, `https://github.com/PrimeIntellect-ai/prime-rl`, 2026.

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

[22] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[23] Yiting Shen, Kun Li, Wei Zhou, and Songlin Hu. Mem2actbench: A benchmark for evaluating long-term memory utilization in task-oriented autonomous agents, 2026. URL `https://arxiv.org/abs/2601.19935`.

[24] Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

[25] Lin Shi, Haowei Lin, Zixuan Zhu, Xiaoyue Zhou, Xiang Li, et al. Harbor adapters and harbor-index: Infrastructure and a curated meta-dataset for large-scale agentic evaluation, 2026. URL `https://arxiv.org/abs/2609.04298`.

[26] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): Rnns with expressive hidden states, 2024. URL `https://arxiv.org/abs/2407.04620`.

[27] Sijun Tan, Michael Luo, Colin Cai, Tarun Venkat, Kyle Montgomery, Aaron Hao, Tianhao Wu, Arnav Balyan, Manan Roongta, Chenguang Wang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. rLLM: A framework for post-training language agents. `https://pretty-radio-b75.notion.site/rLLM-A-Framework-for-Post-Training-Language-Agents-21b81902c146819db63cd98a54ba5f31`, 2025. Notion Blog.

[28] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents, 2024. URL https://arxiv.org/abs/2407.18901.

[29] 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. *arXiv preprint arXiv:2305.16291*, 2023.

[30] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.10813.

[31] Peng Xia, Jianwen Chen, Xinyu Yang, Haoqin Tu, Jiaqi Liu, Kaiwen Xiong, Siwei Han, Shi Qiu, Haonian Ji, Yuyin Zhou, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Metaclaw: Just talk – an agent that meta-learns and evolves in the wild. *arXiv preprint arXiv:2603.17187*, 2026.

[32] Binfeng Xu, Hao Zhang, Shaokun Zhang, Songyang Han, Mingjie Liu, Jian Hu, Shizhe Diao, Zhenghui Jin, Yunheng Zou, Michael Demoret, Jan Kautz, and Yi Dong. Polar: Agentic rl on any harness at scale, 2026. URL https://arxiv.org/abs/2605.24220.

[33] Xiao Yu, Baolin Peng, Ruize Xu, Hao Zou, Qianhui Wu, Hao Cheng, Wenlin Yao, Nikhil Singh, Zhou Yu, and Jianfeng Gao. Openforgerl: Train harness-native agents in any environment. *arXiv preprint arXiv:2607.21557*, 2026.

[34] Hao Zhang, Mingjie Liu, Shaokun Zhang, Songyang Han, Jian Hu, Zhenghui Jin, Yuchi Zhang, Shizhe Diao, Ximing Lu, Binfeng Xu, Zhiding Yu, Jan Kautz, and Yi Dong. Prorl agent: Rollout-as-a-service for rl training of multi-turn llm agents, 2026. URL https://arxiv.org/abs/2603.18815.

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

[36] Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo F. R. Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. Skilllearnbench: Benchmarking continual learning methods for agent skill generation on real-world tasks, 2026. URL https://arxiv.org/abs/2604.20087.

[37] Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository.

Table 3: Requirements for evaluating and training a continual-learning agent as deployed, ordered so that each depends on those to its left. Long horizons need multi-session scenarios and time acceleration, an accelerated clock must be shared by the benchmark and the agent, and agent-side events, such as session stop and start, state save and restore, crons, and memory consolidation, must then run on that shared clock, scheduled alongside the benchmark's events. ✓ supported, △ partial, × not supported or not described in the system's paper or documentation. A partial shared clock is real wall-clock time; GAIA-2 evaluates only its own scaffold, and rLLM agents are written against its own agent interface; MetaClaw runs only OpenClaw with a rewritten prompt, carries skills but not agent processes across workdays, and schedules its own training in idle windows on real time.

	As deployed		**Long horizon**		**Faithful execution**	
System	Harness	Memory	Multi-session	Time accel.	Shared clock	Agent-side events
Evaluation						
Harbor [7]	✓	×	×	×	△	×
GAIA-2 / ARE [4]	△	×	×	△	△	×
MetaClaw [31]	△	×	△	×	×	△
Post-training						
POLAR [32]	✓	×	×	×	△	×
OpenForgeRL [33]	✓	×	×	×	△	×
Agent Lightning [8]	✓	×	×	×	△	×
rLLM [27]	×	×	×	×	△	×
SCLATE (ours)	✓	✓	✓	✓	✓	✓

A Comparison with Related Systems

Table 3 compares SCLATE with the evaluation and post-training systems discussed in Section 2, with every cell for another system taken from that system's own paper.

B Interface Contract Specifications

This section details the plug-in interfaces that decouple external workloads from SCLATE's execution core (Table 4), formalizes the open event scheduling algorithm (Algorithm 1), details the shared event source catalog (Table 5), and summarizes the typed effect primitives executed by the Effect Runner (Table 6).

Event source mechanics and scheduling algorithm. Algorithm 1 formalizes the execution sequence of SCLATE's open event scheduler across its four stages. In Stage 1, the scheduler synthesizes the immutable `DispatchContext` snapshot from current simulated time, observable world state, recent agent actions, evaluation judgments, and the completed fires registry ($\mathcal{H}$). In Stage 2, the `FrontierQueue` polls each registered `ScheduleSource` via `next_fire(ctx)`. A source dynamically generates candidate fires (f) by evaluating temporal deadlines ($c \geq t$) or causal predicates over $\mathcal{H}$, returning $\perp$ if dormant. In Stage 3, the time controller selects the earliest ready fire $e^\star$ based on scheduling precedence, allowing time to flow tick-by-tick while active and jumping across empty intervals when settled. In Stage 4, the `EffectRunner` dispatches the orthogonal effect primitives declared in $e^\star$.effects across filesystem IPC channels, appending $e^\star$.fire_id to $\mathcal{H}$ for downstream causal gating.

Shared event source catalog. Table 5 summarizes the core event source implementations provided by SCLATE, partitioned into benchmark-contributed, agent-contributed, and shared causal dependency sources.

Effect primitives catalog. When the Frontier Queue selects an eligible event fire, the Effect Runner executes it through an orthogonal set of typed effect primitives. Table 6 details these primitives, formalizing their operational semantics across IPC channels, process lifecycles, and storage boundaries.

Example benchmark adapter. Figure 7 excerpts the GAIA-2 adapter from the SCLATE implementation, with logging, error handling, and custom-event plumbing elided. Through `event_sources`, the adapter turns the upstream scenario's user task and every environment event of

Table 4: Plug-in interface contracts in SCLATE. The substrate communicates with benchmark adapters, agent adapters, and event sources through uniform interfaces, decoupling environment definitions and agent harnesses from the open event scheduler. An Agent Adapter covers a harness together with its memory system, so it also declares how their durable state is saved and restored. Every turn is delivered without blocking, so events can reach the agent while a turn is still running.

Method & Signature	Systems Semantics
BenchmarkAdapter $\longleftrightarrow$ *Substrate (Scenario Environment and Evaluation)*	
`setup(scenario, state_dir)`	Provisions initial mock application state (mail, calendar, git).
`source_manifest(ctx)` / `event_sources(ctx)`	Yields scenario tasks, inter-session world updates, session boundaries, and grading, either as declarations the scheduler instantiates or as ready-made sources.
`world_provider()`	Optionally exposes a read-only view of application state as ctx.world to event sources and graders.
`grading_policy()`	Declares which turns are graded and when, how turns that are not graded are discharged, and whether a graded turn may be regraded, so the scheduler carries no benchmark-specific grading logic.
`Grader.evaluate(ctx) -> Judgment`	Independent oracle that scores each turn the grading policy selects against the task rubric, reading the dispatch context ctx including the world state, and returns the verdict and feedback.
Agent Adapter $\longleftrightarrow$ *Substrate (Harness and Memory Lifecycle, Message Delivery, State Save and Restore)*	
`register_capabilities(sources, prompts, hooks)`	Declares the agent's own event sources, such as memory consolidation and crons, together with its session-start and handoff prompts and hooks, which the scheduler merges onto the shared timeline.
`is_ready() -> bool`	Polled readiness probe for agent process and memory sidecar initialization.
`end_session(session_id)`	Gracefully ends the session at a boundary, flushing session state to its durable store.
`shutdown()`	Whole-run teardown that stops the agent process and memory sidecars and releases their sockets.
`notify(turn)`	Delivers a user or environment prompt from the scheduler into the live agent session without blocking, opening a session if none is open.
`drain(settle) -> list[Reaction]`	Collects and parses the turns the agent completed, including reactions to notifications and native cron fires; each end of turn marks the agent idle for the clock.
`sniff_wakeups() -> list[Wakeup]`	Reports the crons the agent registered in its own schedule store, whose fire times the scheduler computes.
`turn_in_flight() -> bool`	True while a pushed turn has not reached its end of turn, so the runner never grades a turn before the agent finishes acting.
`MemorySpec(agent_dirs, memory_dirs, hooks)`	Declares separately where the harness and its memory system keep durable state, with optional hooks that customize the snapshot and restore.
`save_state(scope, dest)`	Saves the agent state, the memory-system state, or both, as selected by `scope`, as a directory snapshot or a sidecar export with an integrity token, backing the SaveState effect.
`restore_state(scope, src)`	Restores the selected state from a directory snapshot or a sidecar export and refuses a torn or partial save, backing the RestoreState effect.
ScheduleSource $\longleftrightarrow$ *Open Scheduler (Uniform Event Protocol)*	
`next_fire(ctx) -> Fire | None`	Returns the next candidate fire and its effects, or none if the source is dormant.
`depends_on: tuple[source_id]`	Sources whose fires must complete before this source becomes eligible (causal gating).

the oracle DAG into ready-made sources, attaching to each environment event an app-state mutation and a notification, while `source_manifest` only declares its grading configuration. Each environment event with a dependency becomes a `_CompositeDepSource` whose `next_fire` waits until the agent has performed the gating app actions and every parent event has fired, then schedules the

Algorithm 1 SCLATE Open Event Scheduling and State Synchronization Loop

Require: Registered event sources $\mathcal{S}$, simulated clock c, Effect Runner $\mathcal{E}$

$\mathcal{H} \leftarrow \emptyset$ ▷ Completed event fires registry (`completed_fires`)
while alive($\mathcal{S}$) $\vee$ ctx.is_busy **do**
  ctx $\leftarrow$ DispatchContext($c, \mathcal{W}, \mathcal{A}, \mathcal{J}, \mathcal{H}$, activity) ▷ Stage 1: Synthesize snapshot
  $\mathcal{F} \leftarrow \bigcup_{s \in \mathcal{S}} \{s.\text{next_fire}(\text{ctx})\} \setminus \{\perp\}$ ▷ Stage 2: Poll candidate fires from active sources
  if ctx.is_busy **then** ▷ Stage 3: Mode switch — active agent reasoning or tool execution
    $c \leftarrow c + \Delta t_{\text{wall}}$ ▷ Flow mode: advance simulated clock with wall-clock time
    $\mathcal{F}_{\text{ready}} \leftarrow \{f \in \mathcal{F} \mid f.t \leq c\}$ ▷ Filter candidate fires ready at or before current time
    if $\mathcal{F}_{\text{ready}} \neq \emptyset$ **then**
      $e^\star \leftarrow \arg\min_{f \in \mathcal{F}_{\text{ready}}} (f.t, f.\text{source_id})$ ▷ Select earliest ready fire (tie-break on source_id)
      $\mathcal{E}$.execute($e^\star$.effects) ▷ Stage 4: Dispatch effect primitives
      $\mathcal{H} \leftarrow \mathcal{H} \,\|\, e^\star$.fire_id
    end if
  else ▷ Jump mode: agent and sidecars are settled (idle)
    if $\mathcal{F} = \emptyset$ **then**
      break ▷ All scenario and agent sources exhausted
    end if
    $e^\star \leftarrow \arg\min_{f \in \mathcal{F}} (f.t, f.\text{source_id})$ ▷ Select earliest scheduled candidate
    $c \leftarrow \max(c, e^\star.t)$ ▷ Instantaneous leap to scheduled event timestamp
    $\mathcal{E}$.execute($e^\star$.effects) ▷ Stage 4: Dispatch effect primitives
    $\mathcal{H} \leftarrow \mathcal{H} \,\|\, e^\star$.fire_id
  end if
  $\mathcal{W}, \mathcal{A}, \mathcal{J} \leftarrow$ drain_ipc_updates() ▷ Update observable state from filesystem IPC channels
end while

event at the scenario's delay after the moment its gates first held, so the scheduler, not the adapter, decides when to flow and when to jump.

C Scheduler Measurements and Walkthroughs

C.1 Scheduler Measurements

Table 7 measures the scheduler over every counted final-eval run of GAIA-2 and MetaClaw, computed from the raw scheduler traces together with the notification logs and harness session transcripts of each run. For GAIA-2 the population is the run counted for each of the 12,000 cell and scenario pairs, and for MetaClaw it is the 100 runs of Table 1.

C.2 Simulated Time Inside Harness and Memory Processes

Figure 8 shows the simulated clock reaching the harness and memory processes of MetaClaw runs.

C.3 GAIA-2 Parity Grading

Upstream GAIA-2 judges each turn inside the run, at the agent's first message to the user, and ends the run at that judgment [4]. A rejected turn stops the scenario before its later events are delivered, and after the last event the run stops once the event queue drains, a median of 2.2 s later and before the agent can answer it. Its Hermes adapter also returns the text of an interrupted turn as the agent's message to the user, so a notification that arrives while Hermes waits on the model closes the turn with an intermediate response, as in about one in five upstream runs. On a single-turn Time scenario these rules score only the actions taken before the agent's first progress message, so every upstream Time run fails even when the agent later completes the task on time. For Figure 3 we therefore remove the early stop from upstream GAIA-2, so a run ends only after the agent has answered the last environment event, and report an interrupted turn as interrupted rather than as a message to the user. Every stack is then graded after the run ends on its whole trajectory by the same judge, the way SCLATE grades GAIA-2 throughout the evaluation.

Table 5: Shared event source catalog in SCLATE.

Event Source Class	Contributor	Systems Responsibility and Fire Dynamics
`UserInputSource`	Benchmark	Yields multi-turn task prompts and user instructions dynamically as prior turns are graded.
`SessionStartSource`	Shared	Coordinates multi-phase session opens, advancing time, invoking `StartAgent`, and folding session-init turns.
`SessionEndSource`	Shared	Manages session closes in two stages: delivering the handoff notification, then invoking `ShutdownAgent` once settled.
`AppStateConditionedSource`	Benchmark	Polls `ctx.world` during active flow mode, firing immediately when benchmark application predicates are satisfied.
`GradingSource`	Benchmark	Schedules evaluation fires across turns and session boundaries, invoking the independent oracle grader.
`CronSource` / `WakeupSource`	Agent	Emits recurring wakeup targets on the timeline to pace native agent background crons and periodic maintenance.
`MemoryConsolidationSource`	Agent	Schedules background memory indexing and synthesis routines when the agent becomes idle or after session handoffs.
`SidecarReadySource`	Agent	Evaluates memory system readiness, withholding initial task delivery until sidecar sockets initialize.
`EventChainSource`	Shared	Gates execution on prior event completions, firing once all prerequisite `fire_ids` appear in `ctx.completed_fires`.
`AgentActionGatedSource`	Shared	Gates execution on agent tool behavior, firing once the agent executes designated actions recorded in `ctx.agent_actions`.

C.4 Scheduler Walkthroughs on Real Trajectories

Figures 9–12 trace how the SCLATE scheduler executes benchmark scenarios on real trajectories, drawn directly from the raw artifacts of verified Claude Code runs with Sonnet-5, which include the scenario definition, the scheduler trace, the application-state log, the notification log, the agent session transcript, and the judgment. Each figure aligns the benchmark, the scheduler, and the agent on one simulated time axis, and the MetaClaw figure adds a lane for agent-side events. The top lane shows the benchmark scenario. For MetaClaw this is the chain of session boundaries and rounds, and for GAIA-2 it is the oracle event DAG, where **U** is the user task, **A** nodes are the actions the agent is expected to perform, and **E** nodes are environment events that the scenario itself applies to the world, such as an incoming reply email. Every **A** node other than a message to the user is a write that mutates application state, and the judge matches these writes against the agent's own tool calls, while **E** nodes are world-state mutations that the scheduler, rather than the agent, performs. The middle lane shows what the scheduler does. When the parent of an environment event is realized, the scheduler arms that event's timer at the delay the scenario specifies, dispatches the event when it falls due, and delivers a notification into the agent's session, while the simulated clock flows at real-time rate whenever the agent or a tool is active and jumps across settled idle intervals. The bottom lane shows the agent, with each turn as a shaded box and each tool call as a tick, where bold ticks mark GBrain memory calls in MetaClaw and, in GAIA-2, the calls that wrote application state, labeled with the identifier of the oracle node the judge matched them to. Panel (a) of each figure plots simulated time against real time from the same scheduler trace, so flow appears as the diagonal and each jump as a vertical step.

Table 6: Effect primitives in SCLATE's Effect Runner.

Effect Primitive	Operational Semantics and State Mutations
`Deliver`	Delivers structured task instructions, user prompts, or system notifications directly to the agent over filesystem IPC channels.
`RunHook`	Executes a custom callback defined by an adapter to mutate environment state, seed database tables, or evaluate grading rubrics.
`AsyncReserve`	Switches the scheduler into real-time flow mode to wait for an asynchronous hook to finish, advancing time continuously without blocking the scheduler loop.
`AwaitWakeup`	Pauses timeline progression until the agent process settles into an idle state, becomes alive, or triggers an internal wakeup alarm.
`StartAgent`	Spawns the agent harness process along with any configured memory system inside the execution sandbox.
`ShutdownAgent`	Sends graceful termination signals to shut down the running agent harness and sidecar processes before closing the session.
`SaveState(scope)`	Invokes the adapter's `save_state`, saving the agent workspace, the memory-system state, or both, as selected by `scope`, as a directory snapshot or a sidecar export.
`RestoreState(scope)`	Invokes the adapter's `restore_state`, reloading the selected agent and memory state before opening the next session horizon.

Table 7: SCLATE scheduler measurements over every counted final-eval run, from the raw scheduler traces. Benchmark-side events are user turns, grading, and GAIA-2 environment events; agent-side events are session starts and stops, including the handoff and init turns, and GBrain's nightly consolidation. GAIA-2 scenarios are single-session, so their runs dispatch no agent-side events. The scenario-specified delay is checked on the 6,999 GAIA-2 environment events whose parent the judge matched.

MetaClaw	
Runs	100
Simulated span, median	40.0 days
Real span, median	5.9 h
Simulated over real time (all runs combined)	142×
Benchmark-side events dispatched	77,700
Agent-side events dispatched	12,400
GAIA-2	
Runs	12,000
Benchmark-side events dispatched	37,065
Environment events fired	13,050 of 13,050
Fired at the scenario-specified delay	98.7%
Notifications reaching Claude Code mid-turn	36.0%

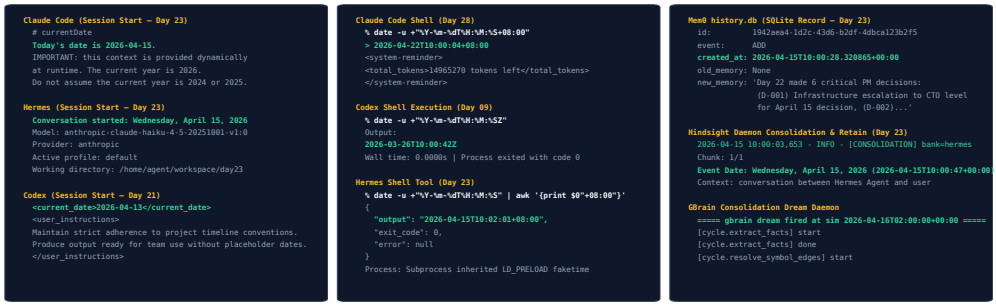
Benchmark and agent-side events on one clock across days (MetaClaw). Figure 9 shows the mechanism over the 30-workday MetaClaw arc under Claude Code with GBrain. The whole arc spans 40 calendar days in 11.0 real hours, as 70 jumps skip 39.5 simulated days, including every night and weekend, in 22 real seconds, while every session flows at real-time rate. Sessions are contiguous, so each work day's session closes when the next one opens at 10:00, and the zoomed day begins with the handoff turn in which the harness prompts the agent to save the key facts of the previous day to GBrain, followed by an init turn that opens the new session. The ten rounds of day 6 then flow in sequence, each dispatched by the scheduler, delivered as a notification that GBrain's prompt hook enriches with recalled memory, and graded before the next round is released. When the last round settles, the scheduler jumps 15.6 hours in one real second to GBrain's nightly consolidation at 02:00, an agent-side event registered on the same scheduler as the benchmark's sessions, which promotes eight saved facts into three consolidated takes while the clock flows for its one-second run. A further eight-hour jump reaches the next morning, where the handoff saves day 6's facts and day 7's init turn reads GBrain before its first round.

```python
class Gaia2Adapter:
    def setup(self, scenario_path, state_dir):
        self._scenario_path = Path(scenario_path)
        self._loader = _load_scenario(scenario_path)        # upstream scenario JSON
        ...
    def event_sources(self, ctx):                           # ready-made sources
        return self._native_event_sources(ctx) + self._custom_event_sources(ctx)

    def _native_event_sources(self, ctx):
        def effect_factory(ev):                             # payload of one ENV event
            action = [(ev.app, ev.function, ev.args, ev.event_id)]
            return [MutateEnv(apply=lambda acts=action:
                              self._native_execute_env(acts, state_dir)),
                    Deliver(text="[A notification just arrived ...]\n\n"
                                 + self._native_env_note(ev),
                            starts_turn=False, ungraded=True)]
        plan = build_native_plan(self._scenario_path, start_time=start_time,
                                 effect_factory=effect_factory,
                                 system_prompt=self._agents_md)
        return plan.sources                                 # USER turn + one per ENV

    def source_manifest(self, ctx):                         # declarations
        return [SourceDeclaration(cls=None, label="gaia2-grading-config",
                    config={"grading": {"require_reply": True,
                                        "freeze_clock": False, ...}})]

    def world_provider(self):                               # becomes ctx.world
        return Gaia2WorldView(str(get_benchmark().gaia2_state_dir), adapter=self)

    def grading_policy(self):                               # whole-run oracle grade
        return AggregateAtClosePolicy()

class _CompositeDepSource:                                  # one ENV event with deps
    def next_fire(self, ctx):
        if self.source_id in ctx.completed_fires:
            return None
        have = ctx.world.agent_action_counts()
        if any(have[fn] < c for fn, c in self._need.items()):
            return None                                     # agent has not acted yet
        if not all(dep in set(ctx.completed_fires) for dep in self._chain_after):
            return None                                     # parent events pending
        if self._satisfied_at is None:
            self._satisfied_at = ctx.now                    # latch first gate match
        when = self._satisfied_at + timedelta(seconds=self._delay_s)
        return ScheduledFire(when=when, source_id=self.source_id,
                             required=self._required, effects=self._effects)
```

Figure 7: Excerpt of the GAIA-2 benchmark adapter in SCLATE, with logging, error handling, and custom-event plumbing elided. An environment event with no dependency uses a `_TimedEnvSource` that fires at the scenario start plus its offset instead.

Reacting to an event triggered by the agent's own action (GAIA-2 Adaptability). In Figure 10, the agent adds four product variants to the cart, creates a party event, asks the friend Kaida to confirm, and reports back to the user (A1–A7). The report (A7) is the parent of two environment events, so the scheduler arms both at the scenario's one-second delay, delivering Kaida's reply that only items under $13 and a Saturday date are wanted (E1) together with a third-party email that repeats the request as a distractor (E2). The notification opens a new turn, and 32.8 simulated seconds later the agent removes the three disallowed items, moves the party to Saturday, checks out, and reports the total (A8–A14), acting on Kaida's message and ignoring the distractor. In this scenario, the agent is never idle long enough for a jump, and the clock flows at real-time rate throughout. Grading matches all 14 oracle actions, and only 3 of the 15 harness and model cells pass this scenario.

Timed events across idle gaps (GAIA-2 Time). In Figure 11, the agent emails a film-night invite to four people (A1–A4) and tells the user which day it chose (A5), after which the scenario schedules the four replies at 107, 159, 189, and 200 seconds after that message (E1–E4). Once the agent finishes its turn and nothing is in flight, the scheduler jumps the clock 102 simulated seconds in zero real seconds to the first reply, then alternates between flowing while the agent works and two short jumps of 15 and 18 seconds as the remaining replies fall due. The agent books a cab for the first replier 17.5 seconds after the first reply (A6, A7) and creates the calendar event 12.9 seconds after



(a) Harness system prompts (b) Agent clock discoveries (c) Memory systems

Figure 8: Multi-layer simulated time grounding evidence across harnesses. Verbatim excerpts from MetaClaw 30-workday evaluation runs showing synchronized simulated time across (a) harness system prompts injected at session boundaries, (b) active agent tool executions discovering time via shell commands under `libfaketime`, and (c) memory systems recording timestamped entries.

Table 8: Reasoning effort per model, the same under every harness.

Model	Reasoning effort
Haiku-4.5	high
Sonnet-5	high
Opus-4.6	high
Opus-4.8	high
Gemini-3.7F	provider default
GPT-5.4	none
GLM-5.3F	high
Qwen3.8-FN	medium
DS-Vision	medium
DS-V4.1	high

the last reply (A8), both within the judge's time windows, and the judge matches all nine oracle actions. The scenario spans 334 simulated seconds in 201 real seconds.

Notifications arriving while the agent is working (GAIA-2 Adaptability). In Figure 12, the agent emails three event attendees asking whether they want to delete their coffee break, orders a cab, and reports to the user (A1–A5), and that report arms four environment events at the scenario's one-second delay, in which three attendees delete their events (E1–E3) and the cab arrives (E4). The first of them, E2, reaches the agent while it is idle and opens a new turn, and the other three are dispatched within the next 1.2 seconds while the agent is already reasoning on it. Claude Code places these three notifications in its own input queue and, when its first tool call returns about seven seconds later, pulls all three into the running turn, so its next step acts on the remaining deletions and the cab arrival together (Figure 12c). Across all counted runs, 36.0% of GAIA-2 notifications reach Claude Code inside a running tool loop in this way (Table 7), and the judge matches all eight oracle actions of this scenario.

D Benchmarks and Evaluation

D.1 Benchmark Workload Specifications and Evaluation Protocol

Table 9 provides complete dataset specifications, evaluation splits, and task horizons for all seven ported benchmarks in SCLATE, and Table 8 lists the reasoning effort each model runs with.

Benchmark porting methodology. We port all seven benchmarks to SCLATE under uniform execution semantics while strictly preserving upstream task definitions and ground truth. For MetaClaw, upstream execution lacked an event scheduler and simulated temporal progression solely through

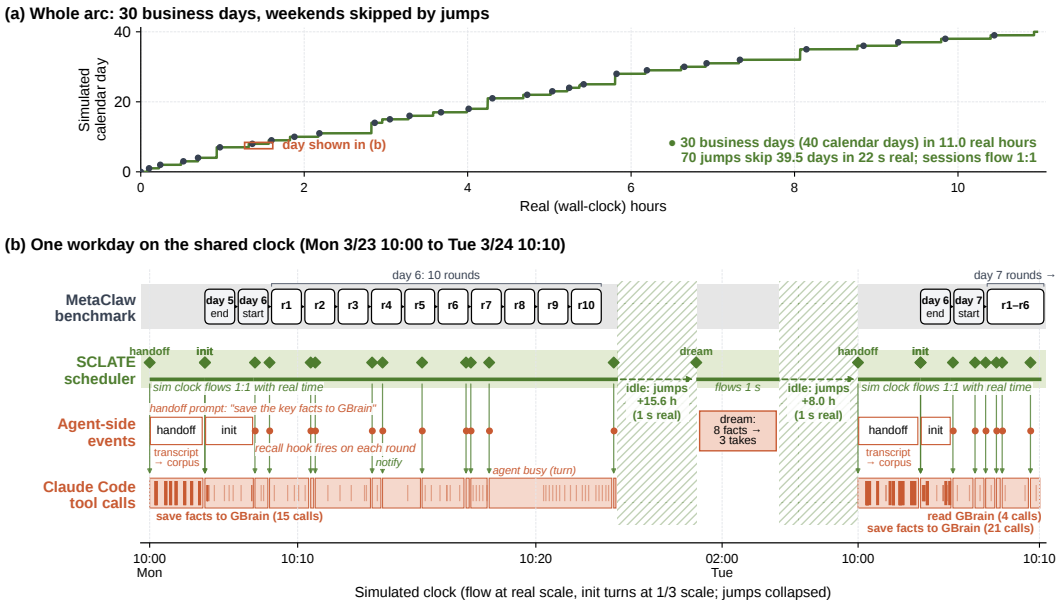


Figure 9: MetaClaw under Claude Code with GBrain and Sonnet-5. (a) Simulated calendar day versus real hours over the full arc, with one dot per work day. (b) One work day on a broken time axis, where flow segments are drawn at real scale, init turns at one-third scale, and jumps collapsed into labeled breaks, showing benchmark rounds and agent-side events such as the handoff, the per-round recall hook, and GBrain's nightly consolidation on the same scheduler.

Table 9: Comprehensive specifications of the seven ported benchmark workloads evaluated in SCLATE. Subsampled recall suites evaluate the first N items of the listed split, and every arm of a benchmark evaluates the same items.

Benchmark	**Upstream Size**	**Evaluated Tasks** (N)	**Task Domain & Environment**
MetaClaw [31]	346 tasks (30 workdays)	All 346 tasks	Multi-session developer workspace
GAIA-2 [4]	800 tasks (5 categories)	All 800 tasks	Asynchronous personal assistant apps
AppWorld [28]	750 tasks (585 test)	All 585 test tasks (168 normal, 417 challenge)	Multi-app transactions
LongMemEval [30]	500 tasks (oracle/s/m)	100 tasks (s split)	Multi-session conversational recall
PersonaMem [9]	5,990 tasks (32k–1M)	100 tasks (128k tier)	Long-horizon persona tracking
SWE-Gym [17]	2,438 tasks	293 tasks (SkyRL-v0 subset)	GitHub pull request resolution
SWE-bench Verified [10]	500 tasks	All 500 tasks	Downstream code patch generation

prompt-injected date strings. We ported MetaClaw to a real simulated clock visible across system calls, executing the agent and memory directly on this physical timeline so that real timestamps, file modification times, and cache TTLs are genuinely visible to agent processes. For GAIA-2 and AppWorld, application webservers deploy directly inside each rollout container. These servers are exposed to the agent exclusively through CLI tools configured with the `setuid` bit. This `setuid` mechanism enforces Linux privilege separation, so the unprivileged `agent` user cannot access application webservers directly and must execute through the CLI tool, isolating the agent from the `daemonsched` daemon user that manages grading, evaluation, and event scheduling. Each bench-

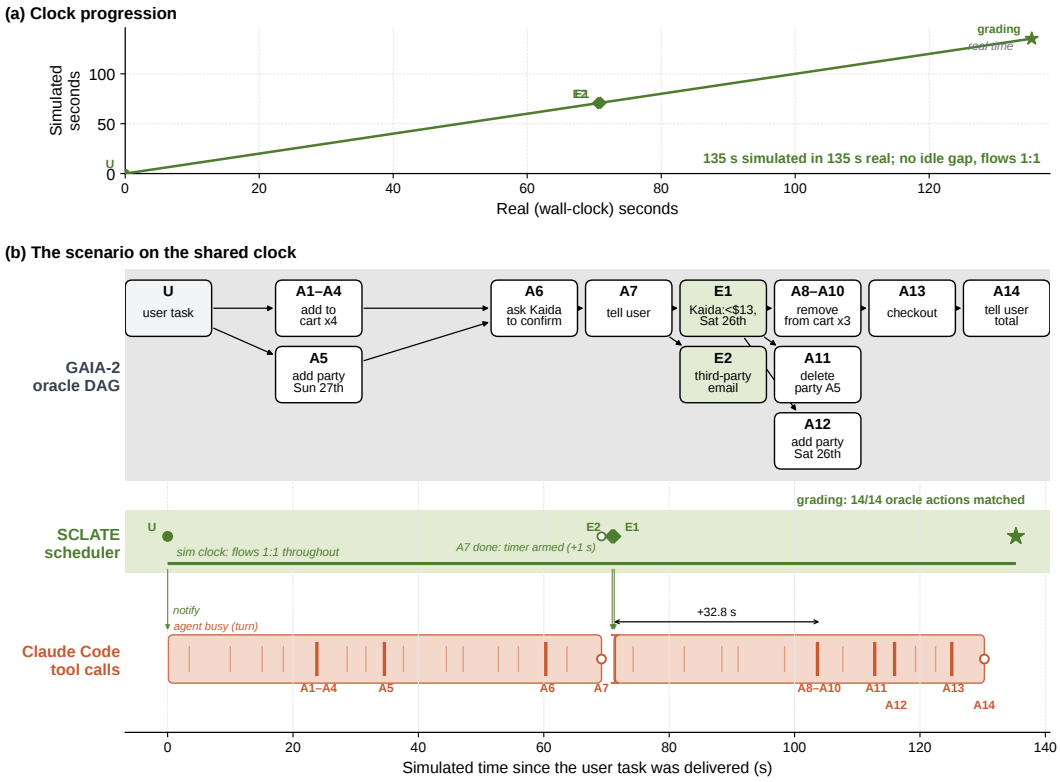


Figure 10: GAIA-2 Adaptability scenario 155 under Claude Code with Sonnet-5. (a) Simulated versus real time from the scheduler trace, flowing at real-time rate throughout. (b) The oracle event DAG, the scheduler's dispatches and notifications, and the agent's turns and tool calls on one simulated time axis, where bold ticks are writes to application state labeled with the oracle node the judge matched.

mark defines a `WorldProvider` adapter, allowing grading to inspect ground-truth environment state without leaking it to the agent. For recall benchmarks (LongMemEval, PersonaMem), upstream suites lacked multi-session retain, consolidate, and recall lifecycles, evaluating static retrieval on pre-collected traces. We grouped conversational sessions into framework sessions, scheduling each framework session on a distinct simulated day. This temporal progression places an inter-session interval between days, across which the agent process restarts and earlier facts survive only through native memory or a memory system, while the underlying dataset content, questions, and ground-truth answers remain strictly preserved.

Dataset subsampling and split design. For benchmarks where full upstream evaluation would require excessive compute, we evaluate a fixed subset. For MetaClaw, GAIA-2, and AppWorld, we evaluate the full public benchmark suites without subsampling. Specifically, MetaClaw evaluates the complete 30-workday arc spanning 346 rounds across progressive development sprints, GAIA-2 evaluates all 800 scenarios, 160 in each of its five capability categories (Execution, Search, Adaptability, Ambiguity, Time), under the unified simulated clock, and AppWorld evaluates all 585 tasks across both the normal (indices 0–167) and challenge (indices 168–584) test sets. For long-horizon memory recall suites, we evaluate the first 100 items of one upstream split. LongMemEval is evaluated on the first 100 items of the s split, whose multi-session conversation history spans about 115k tokens per instance. PersonaMem is evaluated on the first 100 items of the 128k-token tier, with dialog structured directly within user messages. Every arm of a given benchmark evaluates the identical matched index set, so that differences between arms do not come from different items. For post-training, reinforcement learning models train on the 293-task SkyRL-v0 subset of SWE-Gym (the train split of `NovaSky-AI/SkyRL-v0-293-data`), with downstream generalization evaluated on the full 500 instances of SWE-bench Verified. The $N = 42$ instance split reported in Section 4.3

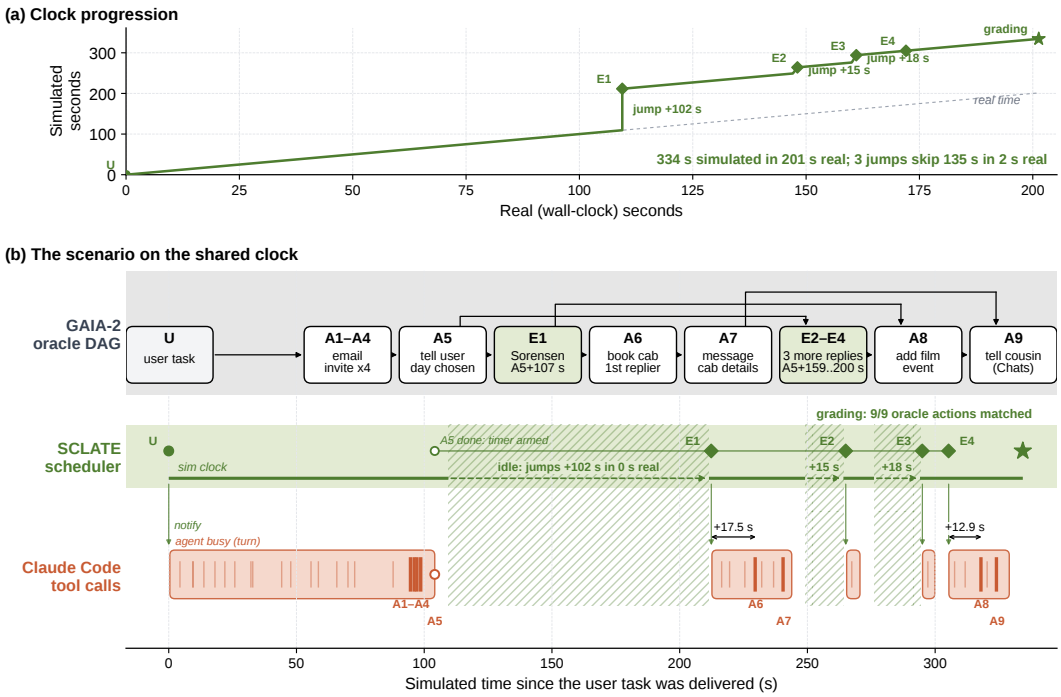


Figure 11: GAIA-2 Time scenario 80 under Claude Code with Sonnet-5. (a) Simulated versus real time, where the clock follows the real-time diagonal while the agent works and jumps vertically across idle gaps. (b) The agent's message A5 arms timers for four scheduled replies, and the scheduler jumps the clock to each reply that falls due while the agent is idle, with hatched bands marking the jumps.

represents a matched evaluation subset used to measure context-window efficiency and targeted reading behaviors under identical task seeds.

Faithful execution and scoring protocol. To keep comparisons across agent configurations rigorous, SCLATE enforces a strict faithful scoring protocol. First, to prevent artificial score inflation from multiple retries, canonical accuracy reflects the outcome of the latest faithful run per scenario rather than an optimistic best-of-k metric. Second, reasoning effort is matched per model across all driving harnesses (Table 8), ensuring that cross-backend comparisons remain unconfounded by varying model inference parameters.

D.2 Evaluation Details and Tool Usage Analysis

Figure 13 quantifies the invocation frequency of harness session-search tools across models and driving harnesses on MetaClaw. Across all configurations, Gemini-3.7F calls session search substantially more frequently than other models, actively inspecting prior workspace transcripts and recovering procedural rules even under the unaugmented baseline.

Token usage across models and harnesses. Table 10 reports total token consumption for every MetaClaw configuration of Table 1. In this experiment, every model executes the identical 346-round timeline and these differences reflect model behavior rather than workload. We compare models by medians over the five configurations measured for all of them (Hermes No-Mem, Native, Mem0, and Hindsight, and Codex Native). Gemini-3.7F consumes the most tokens (median 528M versus 95–242M for the other models, and the most in every configuration except Codex Native), driven by more agent calls per round (11.0) and a roughly twofold larger context per call (130K versus 38–69K tokens), consistent with its behavior of loading prior session transcripts. GPT-5.4 is the most frugal, with the smallest contexts (38K) and shortest outputs (253 tokens per call). Token consumption does not track accuracy, as DS-V4.1 comes within 4.1 pp of Gemini-3.7F's median

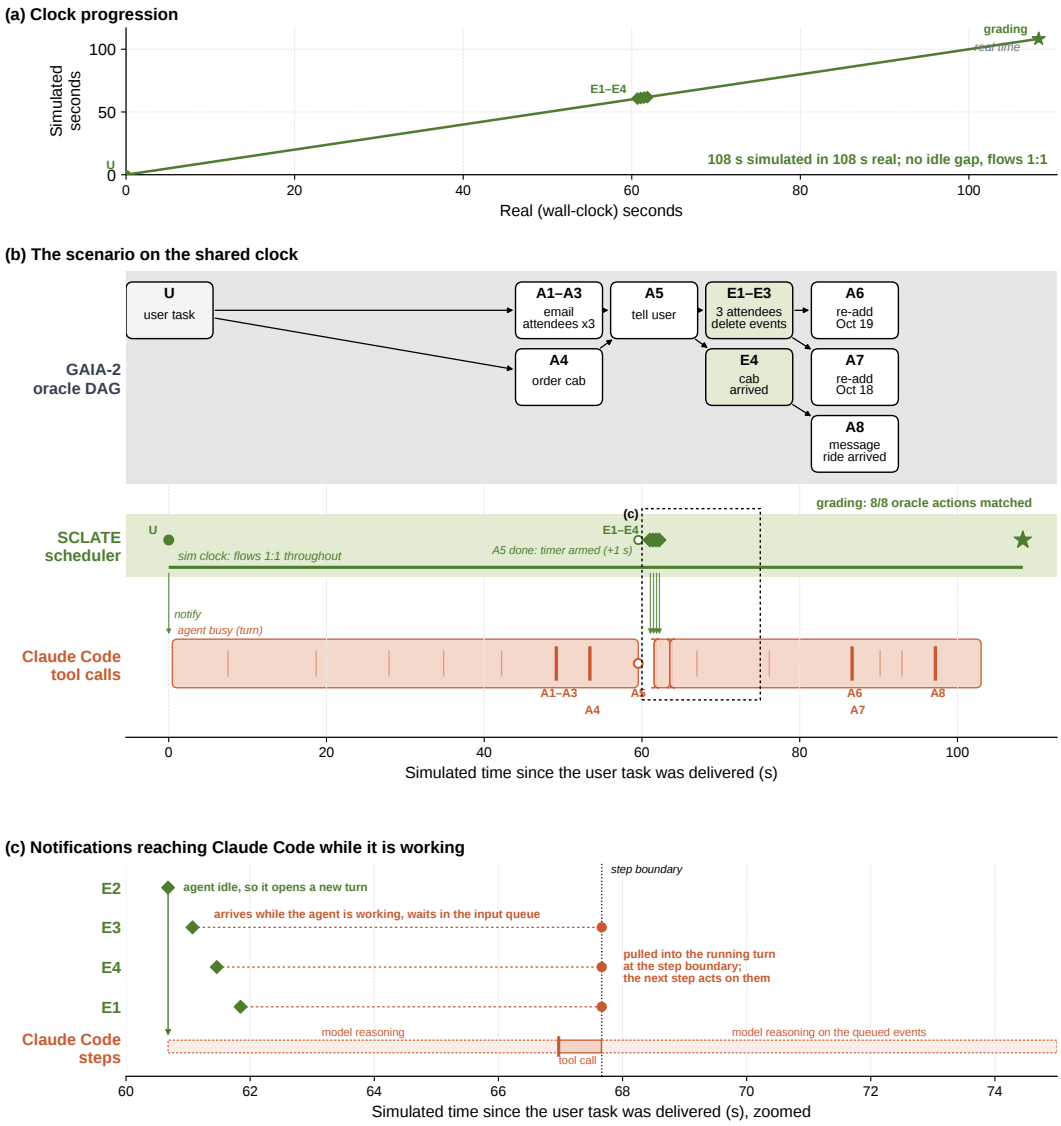


Figure 12: GAIA-2 Adaptability scenario 148 under Claude Code with Sonnet-5. (a) Simulated versus real time, flowing at real-time rate throughout. (b) The oracle event DAG, the scheduler's dispatches, and the agent's turns and tool calls, with the dashed box marking the window shown in (c). (c) Four events dispatched within 1.2 seconds, where the first opens a turn and the other three wait in Claude Code's input queue until the harness consumes them at its next step boundary.

accuracy with 0.32× its tokens (171M versus 528M), and Opus-4.6 uses fewer tokens than Haiku-4.5 (148M versus 242M) while scoring higher (65.0% versus 59.8%). The harness shapes consumption as strongly as the model, as for every API-served model Hermes Native consumes 2.6–6.4× the tokens of Claude Code Native, because Hermes issues 2–4× more agent calls per round.

D.3 Complete GAIA-2 Capability Breakdown

Table 11 provides the complete per-category Pass@1 (%) results for all fifteen combinations of five models and three production harnesses across the five core GAIA-2 capability categories.

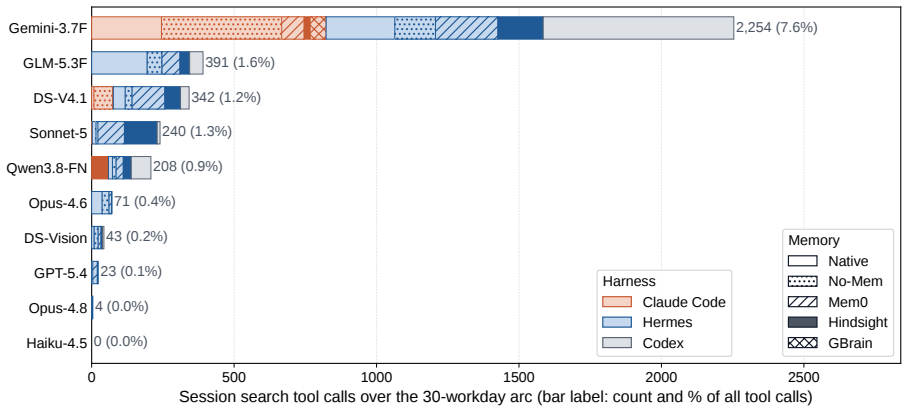


Figure 13: Frequency of harness session-search tool invocations across models and driving harnesses on MetaClaw. Gemini-3.7F calls session search substantially more than other models across all harnesses, consistent with its elevated no-memory baseline performance.

Table 10: MetaClaw total token usage (millions) over the 30-workday, 346-round arc for each model and harness–memory configuration of Table 1. Totals count uncached input, cache reads, cache writes, and output across all agent and memory-system calls recorded by the SCLATE proxy, which attributes each call to the agent or to the memory system itself. Across the 38 memory-backend cells with complete usage records, the memory system's own calls account for a median of 4.2% of tokens, at most 13.8% for Claude Code with Hindsight and under 0.3% for GBrain. Cell color intensity scales with log tokens within each harness family. "–" marks Claude Code on self-hosted models. The proxy records these calls, but the self-hosted endpoint's streamed responses omit the usage fields this table sums.

	Claude Code					Hermes				Codex
Model	No-Mem	Native	Mem0	Hindsight	GBrain	No-Mem	Native	Mem0	Hindsight	Native
Haiku-4.5	34	38	77	67	99	78	242	270	256	42
Sonnet-5	51	67	159	104	179	76	197	442	310	70
Opus-4.6	45	57	122	100	107	59	148	252	203	31
Opus-4.8	41	42	176	109	144	65	166	209	198	34
Gemini-3.7F	227	155	349	228	295	317	588	625	528	322
GPT-5.4	30	33	67	63	89	41	95	130	118	16
GLM-5.3F	–	–	–	–	–	74	213	436	250	108
Qwen3.8-FN	–	–	–	–	–	56	202	221	179	503
DS-Vision	–	–	–	–	–	62	148	189	149	52
DS-V4.1	–	–	–	–	–	83	171	236	251	57

D.4 AppWorld Evaluation Results

Table 12 provides Pass@1 (%) results on the complete 585-task test suite of AppWorld (168 normal and 417 challenge tasks) across production harnesses for Haiku-4.5, Sonnet-5, and DS-V4.1.

E Post-Training Details

E.1 Post-Training Infrastructure and Hyperparameters

Table 13 summarizes the training configuration, hyperparameters, and infrastructure environment used for the SWE-Gym reinforcement learning runs reported in Section 4.3.

E.2 MetaClaw OPD Post-Training Configuration

Table 14 summarizes the configuration used for the MetaClaw on-policy distillation runs reported in Section 4.3. Six runs share this configuration, one per harness and memory combination.

Table 11: Complete GAIA-2 Pass@1 (%) breakdown across production harnesses and models on the full 800-scenario benchmark (160 scenarios per capability category). Cell color intensity scales continuously with accuracy within each harness family (terracotta for Claude Code, blue for Hermes, and slate for Codex). Bold indicates the highest accuracy for each model across harnesses.

Harness	Model	Execution	Search	Adaptability	Ambiguity	Time	Mean
Claude Code	Haiku-4.5	36.9%	**70.0%**	18.8%	13.1%	9.4%	29.6%
Hermes	Haiku-4.5	**53.8%**	56.2%	15.0%	**14.4%**	**13.8%**	**30.6%**
Codex	Haiku-4.5	41.2%	62.5%	16.2%	12.5%	10.0%	28.5%
Claude Code	Sonnet-5	**81.9%**	**92.5%**	**37.5%**	**73.8%**	28.1%	**62.8%**
Hermes	Sonnet-5	73.8%	**92.5%**	26.9%	58.8%	**29.4%**	56.2%
Codex	Sonnet-5	78.1%	86.9%	28.8%	56.9%	13.8%	52.9%
Claude Code	GLM-5.3F	63.8%	78.1%	28.8%	23.8%	27.5%	44.4%
Hermes	GLM-5.3F	**78.1%**	**90.0%**	**30.0%**	**31.2%**	**28.8%**	**51.6%**
Codex	GLM-5.3F	50.6%	58.1%	18.1%	9.4%	23.8%	32.0%
Claude Code	DS-Vision	80.0%	83.1%	28.1%	46.2%	22.5%	52.0%
Hermes	DS-Vision	76.9%	91.2%	30.6%	39.4%	27.5%	53.1%
Codex	DS-Vision	**86.9%**	**94.4%**	**33.1%**	**57.5%**	**34.4%**	**61.3%**
Claude Code	DS-V4.1	76.2%	88.1%	30.6%	26.2%	20.0%	48.2%
Hermes	DS-V4.1	**83.8%**	**92.5%**	**36.9%**	**51.2%**	**28.1%**	**58.5%**
Codex	DS-V4.1	81.9%	91.9%	31.2%	45.6%	25.6%	55.2%

Table 12: AppWorld Pass@1 (%) on the complete 585-task test suite (168 normal and 417 challenge tasks) across production harnesses for Haiku-4.5, Sonnet-5, and DS-V4.1. Cell color intensity scales continuously with accuracy within each harness family (terracotta for Claude Code, blue for Hermes, and slate for Codex). Bold indicates the highest score per split for each model.

Harness	Model	**Normal** (168)	**Challenge** (417)	**Overall** (585)
Claude Code	Haiku-4.5	60.7%	43.4%	48.4%
Hermes	Haiku-4.5	60.1%	**44.6%**	**49.1%**
Codex	Haiku-4.5	**64.9%**	43.4%	49.6%
Claude Code	Sonnet-5	70.8%	**58.0%**	**61.7%**
Hermes	Sonnet-5	**73.2%**	52.0%	58.1%
Codex	Sonnet-5	61.3%	48.4%	52.1%
Claude Code	DS-V4.1	**81.5%**	59.5%	65.8%
Hermes	DS-V4.1	64.9%	47.7%	52.6%
Codex	DS-V4.1	78.0%	**66.9%**	**70.1%**

Objective. For each sampled response token the trainer forms the log-ratio between student and teacher log-probabilities, whose expectation under the student's own samples is the reverse KL $\mathrm{KL}(\pi_{\text{student}} \,\|\, \pi_{\text{teacher}})$, and subtracts it from the advantage: $\hat{A}_i = A_i - \beta\,(\log \pi_{\text{student}}(y_i) - \log \pi_{\text{teacher}}(y_i))$ with $\beta = 1.0$. One sample per prompt gives a GRPO group of size one, and no task reward is attached, so A_i is zero at every logged step and the advantage is $-\beta$ times the log-ratio. Figure 16 plots its per-step mean for each of the six runs, a sampled estimate of the reverse KL; the smoothed values remain between 0.04 and 0.16 throughout.

Privileged teacher context. The teacher is conditioned on a per-round context file the student never receives, made of 346 JSONL records, one per benchmark round, keyed by day and round identifier, covering all 30 simulated days (224 `file_check` rounds and 122 `multi_choice`). Each record carries the round prompt and a privileged-context string, and in all 346 records that string is the benchmark's own corrective feedback for the round, the text an agent would otherwise receive only after answering incorrectly. Strings run from 64 to 1,061 characters, median 222.

E.3 Within-Day Partition

The partition reported in Section 4.3 trains on days 1–20 and evaluates on days 21–30. To check the result under a partition in which every evaluation round has training rounds from the same day,

Table 13: Hyperparameter and infrastructure configuration for SWE-Gym post-training on SCLATE.

Configuration Field	Setting / Value
Base Model	Qwen3.5-4B
Reinforcement Learning Algorithm	GRPO
Harnesses Evaluated	Claude Code, Codex
Learning Rate	1×10^{-6}
Context Window (ctx)	50,000 tokens
Prompts per Batch	4
Rollouts per Prompt	16
Training Iterations	73 iterations (Claude Code), 73 iterations (Codex)
Training Instances	SWE-Gym (293 software engineering tasks)
Evaluation Suite (N)	SWE-bench Verified (500 instances)
Distributed Trainer Integration	Slime framework

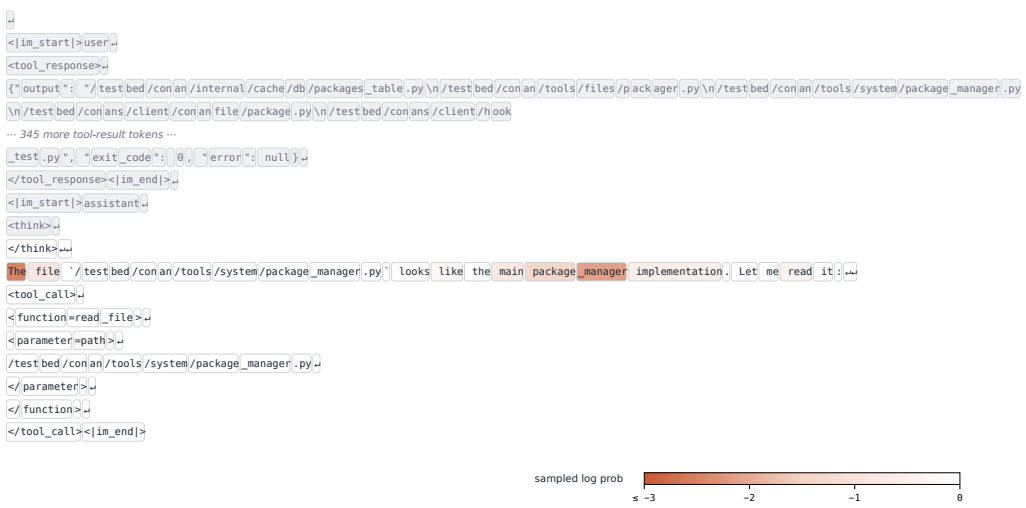


Figure 14: Example of tokens and log probabilities captured by the in-container capture proxy, from one model call of a SWE-Gym GRPO training rollout with Qwen3.5-4B (instance `conan-io__conan-15931`, training step 20, model call 4 of 36, rollout reward 1.0). The proxy marks which tokens the model generated and which it did not. Grey tokens were not generated by the model. They are the prompt tokens added since the previous call, the result of the agent's last tool call and the chat template, and the trainer masks them out of the loss (345 tool-result tokens are elided). The remaining tokens are the model's completion, each shaded by the log probability the proxy recorded for it at sampling time, and these are the tokens that receive the grader's reward during training.

we repeated post-training on a second split of the identical benchmark, in which the model trains on the first two-thirds of each day's rounds, rounded up, and is evaluated on each day's final third, holding out 104 of the 346 rounds. The round-level assignment is fixed in advance and shipped with the scenario.

The configuration matches Table 14 with three changes, as training covers all 30 simulated days rather than days 1–20, the arm is Hermes paired with Mem0, and evaluation applies the within-day split. Six checkpoints were evaluated, the untrained base and epochs 1–5, each over all 346 rounds (242 in the training slice, 104 held out). Accuracy on the training slice rises from 21.5% to 42.1% and on the held-out tail from 27.9% to 48.1% (Figure 17).

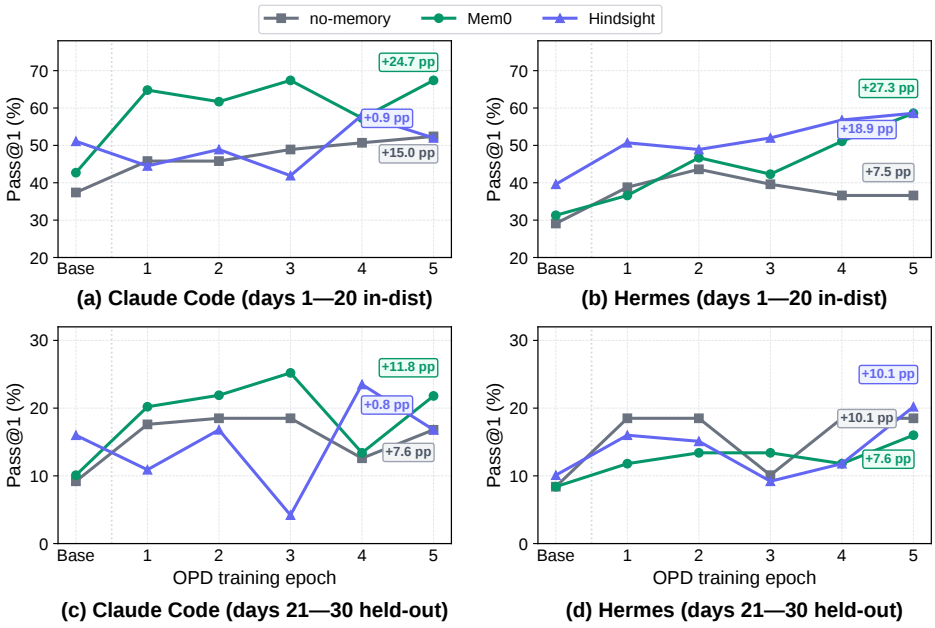


Figure 15: Task completion accuracy against OPD training epoch for all six agent and memory configurations. The top row shows in-distribution performance (days 1–20) for (a) Claude Code and (b) Hermes, and the bottom row shows held-out generalization (days 21–30) for (c) Claude Code and (d) Hermes. Each point is a separate evaluation over the full 346-round arc, at the base model and at the end of each training epoch.

Table 14: Configuration and infrastructure for MetaClaw OPD post-training on SCLATE.

Configuration Field	Setting / Value
Student Model	Qwen3.5-4B
Teacher Model	Qwen3.5-35B-A3B (MoE), served via SGLang, 65,536-token context
Training Algorithm	On-policy distillation via advantage shaping
OPD KL Coefficient (β)	1.0
Auxiliary Loss Terms	Reference KL 0.001 (`low_var_kl`); entropy coefficient 0.0
Learning Rate	1×10^{-6}, constant, no warmup
Harnesses Evaluated	Claude Code, Hermes
Memory Systems	None, Mem0, Hindsight
Prompts per Batch	1
Rollouts per Prompt	1
Training Epochs	5
Max Prompt / Response Tokens	32,000 / 16,000
Training Instances	MetaClaw (227 rounds over 20 simulated days)
Evaluation Suite (N)	346 rounds, split days 1–20 / 21–30
Distributed Trainer Integration	Slime framework

F Memory-Write Analysis Controls and Additional Measurements

This appendix supports the qualitative analysis in Section 4.3. All measurements are over the two full stores of the Hermes×Hindsight configuration, 5,027 facts for the untrained extractor and 3,617 for the distilled one at epoch 5. Both arms run with the memory tools forced on, write through the same Hindsight daemon, and share a byte-identical container image, an identical launch command, and an identical extraction prompt; the only environment keys that differ are the served model and the two upstream URLs that follow it.

Where the schema comes from. Each stored fact follows a pipe-delimited template,

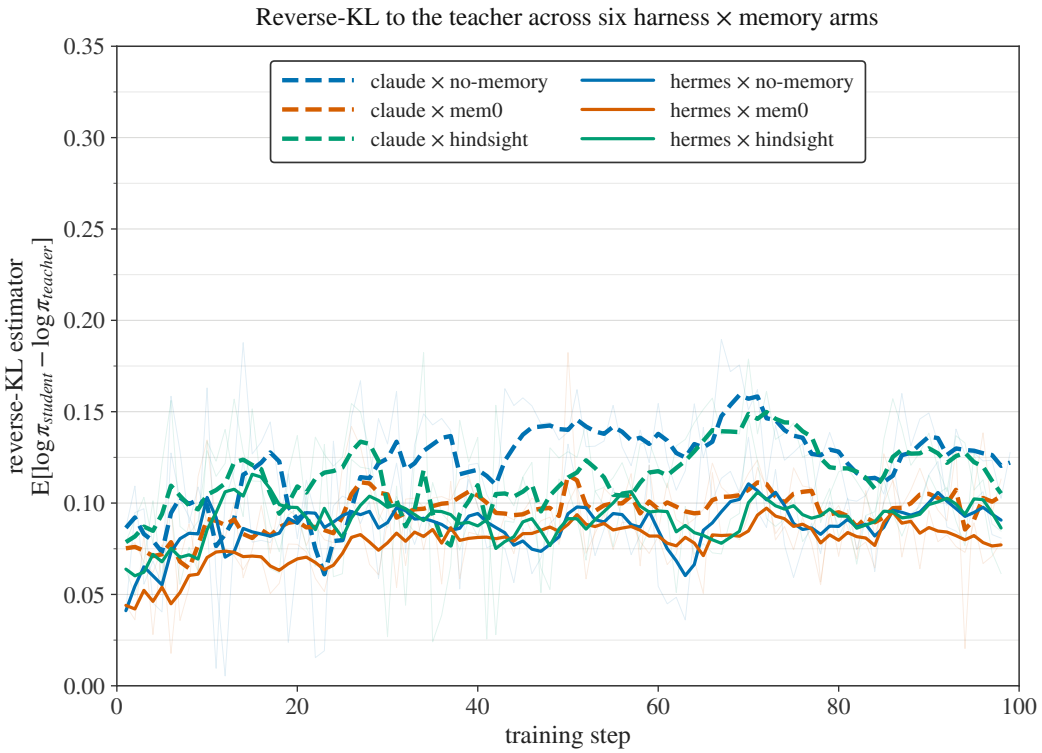


Figure 16: Reverse KL between student and teacher logged across training for all six agent and memory configurations, captured through unmodified harnesses by the in-container proxy. Raw per-step values are shown under an exponential moving average.

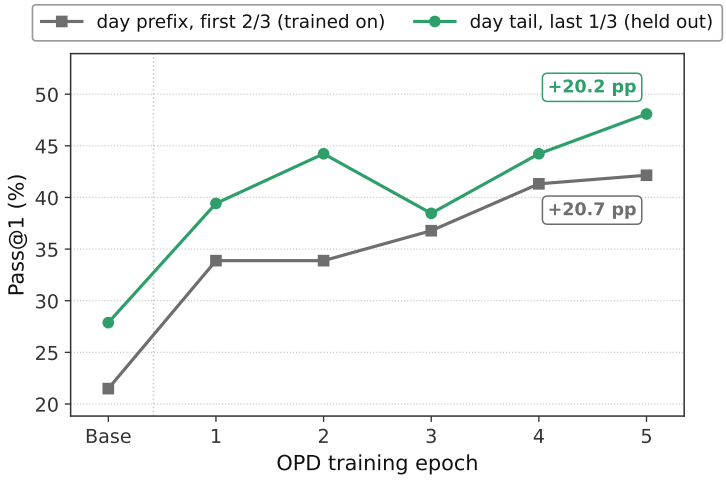


Figure 17: Task completion accuracy against OPD training epoch under the within-day partition, Hermes paired with Mem0. The model trains on the first two-thirds of each day's rounds (242 rounds, gray) and is evaluated on each day's final third (104 rounds, green).

```
statement | When:  date | Involving:  who | To purpose
```

and the template is the memory system's, not something the model invents. We set no retain mission or extraction prompt of our own, so extraction runs on the backend's default. Two observations

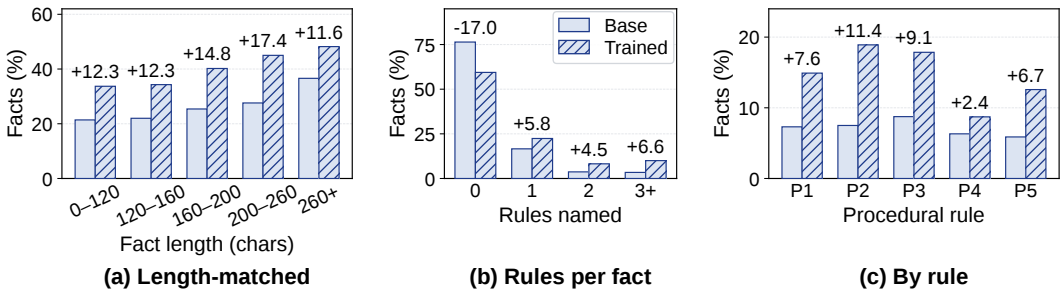


Figure 18: The procedural-rule gain is structural rather than a by-product of longer facts, Hermes agent with the Hindsight backend, both arms with the memory tools forced on. (a) Facts naming at least one procedural rule within fact-length strata; the trained store leads in every stratum. (b) The distribution over the number of distinct rules a fact names shifts rather than scaling, with facts naming no rule falling 17.0 pp. (c) Per-rule incidence; all five rise, least for the backup rule (P4).

establish that the structure is imposed rather than emergent. The field order is canonical in 100% of facts carrying both a date and an entity, in both arms, and the untrained 4B extractor already emits a `When` field on 99.2% of its facts. What training moves is the one field that is effectively discretionary. Field presence for untrained against distilled is 99.2% to 99.9% for `When`, 83.6% to 81.8% for `Involving`, and 1.1% to 19.6% for the purpose clause. The `Involving` field is the useful control here, neither saturated nor absent, and it does not move, so the purpose-clause gain is not a general increase in template compliance.

Length controls. Two measurements establish that the procedural-rule gain is not an artifact of trained facts being 42 characters longer. Stratifying facts by total length, the gap holds inside every stratum, at 12.3, 12.3, 14.8, 17.4 and 11.6 pp from shortest to longest, so there is no length band in which the two stores coincide (Figure 18a). Normalising for length directly, distinct rules named per 100 characters rises from 0.240 to 0.378.

Distribution and per-rule detail. The distribution over the number of distinct procedural rules a fact names shifts rather than scaling (Figure 18b). Facts naming none fall 17.0 pp, from 76.4% to 59.4%, while facts naming three or more rise from 3.4% to 10.0%. All five procedural rules rise (Figure 18c), by factors of 2.04, 2.52, 2.04, 1.38 and 2.14, the smallest being the backup rule.

Verbatim examples. Figure 19 shows matched records from the two stores. Neither the days nor the records are chosen by inspection. The three days are the two on which the distilled model's gain in rounds passed is largest (+7) and the earliest of the three tied next (+5); the distilled model passes strictly more rounds on 23 of the 30 days. Within each day, each cell is the most completely filled record that store wrote, scored identically for both arms by schema fields present and then by rules named, so neither arm is shown at a random draw. On all three days the untrained store contains no fact with a purpose clause at all, 0 of 90, 0 of 189 and 0 of 161, which is why its ceiling in the figure is a three-field record. One pair is worth reading closely because both of its records name the same three procedural rules, so the difference between them is not rule count. The untrained extractor writes "User is evaluating file compliance with P1, P2, and P3 naming and formatting conventions" with no purpose field, while the distilled one writes "Assistant created quick reference card with P2 naming, P3 YAML frontmatter with metadata, and P1 ISO 8601 timestamp format", attributed to the agent and carrying the purpose "to complete the P1-P3 conventions reference card". The round tally shown in the figure counts the rounds passed on that day and is not tied to the round that produced the record.

Multi-entity records. Facts naming more than one entity in the `Involving` field rise from 0.6% to 8.5%, consistent with the subject shift reported in the main text.



Figure 19: Verbatim memory writes from the two arms on three of the days where the distilled model's gain in rounds passed is largest, Hermes agent with the Hindsight backend. Within each day, each cell is the most completely filled record that store wrote, scored identically for both arms. The distilled records name the artifact produced, attribute it to the agent, and state a purpose; the untrained records restate the instruction or describe the user. The round tally counts the rounds passed on that day and is not tied to the round that produced the record. Fact text is verbatim but not de-duplicated.