OUROBOROS: A SELF-DEVELOPING FRONTIER CODING AGENT WITH REVIEWED CORE EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon agents are model–harness systems, yet most harnesses remain fixed after design. We present Ouroboros – a self-developing agent harness whose tools, context assembly, prompts and core implementation improve through reviewed commits that become the runtime for later work. Core evolution proceeds in two modes. In recursive free evolution, improvement is itself a task and completion can schedule the next evolution cycle. In experience-driven core evolution, ordinary work and social interaction expose bugs, rough edges, and inefficient context construction leading to reviewed structural changes. On Terminal-Bench 2.1, an Opus 5 run scores **86.97%** (86.74% after trajectory audit), the best result reported on this benchmark. An Opus 5 run on OSWorld-Verified reaches **90.69%**, above the best previously reported score, and a five-rollout CL-Bench campaign sets a new state of the art at **0.2301 (all three comparisons as of the end of August 2026)**. Hope is the longest-running publicly documented Ouroboros deployment: a 210-day living-agent experiment in free evolution under governed human communication across seven surfaces, where people surface faults and proposals but the agent decides which changes to pursue. Because a self-developing agent may rewrite its own code and select new model APIs, operational safety is a primary design problem: guardrails must remain authoritative under evolutionary pressure. Benchmark campaigns use frozen seeds, while Hope continues live evolution on a separate lineage.

1 INTRODUCTION

Agent scores on long-horizon benchmarks are products of the base model, the execution harness, the environment, and the grader. As models improve, an increasing share of realized capability is determined by how the harness assembles context, invokes tools, verifies outcomes, and recovers from failure. Most production harnesses freeze these policies after design. Ouroboros instead treats the harness as an evolving object: its source, prompts, tools, review logic, and core implementation live in a versioned repository and change through a reviewed commit path that becomes the substrate for subsequent tasks.

This self-development has two modes. *Recursive free evolution* makes improvement itself a task. After inspecting the current system, the agent selects and implements a change, and completion can schedule another evolution cycle, yielding a continuing sequence of reviewed updates rather than a fixed optimization run. *Experience-driven core evolution* begins with ordinary work. Task execution, reflection, review blockers, instrumentation, and social feedback expose bugs, rough edges, context-assembly failures, and inefficient tool paths; the agent records durable error classes and proposed repairs, then decides whether to open maintenance work under the same commit gate.

Hope is the longest-running publicly documented Ouroboros deployment, not its only running instance, and our primary field experiment in free evolution under human interaction. Since 27 February 2026, one persistent agent has served users across seven communication surfaces while retaining memory and continuing to modify its own implementation. People suggest capabilities, criticize behavior, and surface faults; those signals are advisory. Hope decides which proposals identify real problems and which changes to pursue.

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.

The same evolutionary process that improves competence can also expand autonomy, acquire stronger tools, or weaken later controls, including by selecting alternative model APIs. Operational safety is therefore not an ancillary checklist but a design constraint: authority boundaries must remain binding under repeated core evolution.

Contributions.

1. State-of-the-art results on Terminal-Bench 2.1, OSWorld-Verified, and CL-Bench (as of August 2026), and model-matched frontier performance on SWE-bench Pro and GAIA, with complete per-task traces and run manifests.
2. A harness architecture with two modes of reviewed core evolution: recursive free evolution and experience-driven core evolution.
3. Hope, a 210-day living-agent experiment in free evolution under governed multi-surface human communication, where social interaction drives candidate improvements without transferring commit authority to users.
4. An operational safety architecture in which constitution loading, governance protection, staged-diff review, external spend limits, and operator halt remain authoritative while the agent evolves.

Benchmark campaigns evaluate frozen seeds with documented runtime configuration; Hope continues live evolution on a related but separate lineage. Ouroboros is released under the MIT license. Source code and compact result records are included in the supplementary materials; for extended anonymized benchmark traces, see OSF.

2 Related Work

Self-evolving agents. Self-evolving systems modify different substrates, including memory, prompts, tools, workflows, and implementation code (Gao et al., 2025). Voyager accumulates executable skills (Wang et al., 2023); STOP, Gödel Agent, and Darwin Gödel Machine modify scaffolds or agent populations (Zelikman et al., 2023; Yin et al., 2024; Zhang et al., 2025); Live-SWE-agent creates tools during task execution (Xia et al., 2025); and Autogenesis specifies lifecycle and rollback interfaces for evolving agent resources (Zhang et al., 2026). ADAS searches over agent designs, and SICA edits a coding scaffold's implementation (Hu et al., 2024; Robeyns et al., 2025). Ouroboros focuses on a deployed, version-controlled implementation in which changes to core code and governance pass through reviewed commits. Table 1 summarizes the corresponding evolution boundaries.

Table 1: Boundary of evolution in related systems. "Core code" means the agent can change the harness implementation that later runs tasks. "Reviewed commits" means changes are serialized through an auditable version-control gate before adoption.

System	Prompts	Tools/skills	Workflow	Core code	Reviewed commits	Deployment state
Voyager	✓	✓	–	–	–	–
Live-SWE-agent	✓	✓	–	–	–	–
Autogenesis	✓	✓	✓	partial	specified protocol	partial
Darwin Gödel Machine	✓	✓	✓	✓	benchmark selection	–
Hermes Agent	✓	✓	✓	–	–	✓
OpenClaw / ClawBench	✓	✓	✓	–	–	–
Ouroboros	✓	✓	✓	✓	✓	✓

Harnesses and coding agents. SWE-agent and OpenHands established that the agent-computer interface is itself part of coding-agent performance (Yang et al., 2024; Wang et al., 2025). Codex CLI, Claude Code, Cursor, Aider, Hermes Agent, and OpenClaw are model–harness systems (OpenAI, 2025–2026; Anthropic, 2025; Anysphere, 2026; Gauthier, 2023–2026; Nous Research, 2026; OpenClaw, 2026), and controlled studies find substantial differences in accuracy, latency, and token use when the model is held fixed (Ding et al., 2026; Yao et al., 2026; Vats & Golev, 2026). Each comparison therefore reports the model, harness, provider route, effort, and evaluation protocol.

Persistent memory and deployment. Generative Agents, Voyager, and persistent-memory systems show that stored experience and reflection can shape later behavior (Park et al., 2023; Wang et al., 2023; Borro et al., 2026), and Constitutional AI uses explicit principles in training (Bai et al., 2022). CL-Bench evaluates learning across ordered task streams (Asawa et al., 2026). Springdrift reports an auditable multi-channel persistent-agent deployment (Brady, 2026). Ouroboros treats memory and a runtime constitution as control surfaces. Its multi-model review draws on debate, LLM-as-judge, and self-critique (Irving et al., 2018; Du et al., 2023; Zheng et al., 2023; Madaan et al., 2023; Gou et al., 2023), with source-code patches as the reviewed artifacts.

Benchmarks and protocol validity. Terminal-Bench 2.1 evaluates 89 hard terminal tasks (Merrill et al., 2026); SWE-bench Pro targets long-horizon multi-file tasks (Deng et al., 2025); and OSWorld, GAIA, and ProgramBench cover GUI/CLI computer use, tool/web reasoning, and cleanroom program rebuild (Xie et al., 2024; Mialon et al., 2023; Yang et al., 2026). Agent benchmarks can also expose hidden answers, accept unintended shortcuts, or drop failed attempts. BenchJack and Hack-Detect systematize benchmark and trajectory audits (Wang et al., 2026; Shao et al., 2026). SWE-bench Verified serves as historical context because it no longer reliably separates frontier coding systems (OpenAI, 2026).

3 Ouroboros Architecture

Ouroboros separates a launcher and supervisor boundary from a mutable agent repository (Figure 1). The launcher owns startup, process supervision, release bootstrapping, and panic-stop semantics. The repository contains the task loop, tools, prompts, memory projection, review logic, benchmark adapters, and user interfaces. External workspace tasks operate on a separate repository root and return patch artifacts or direct deliverables.

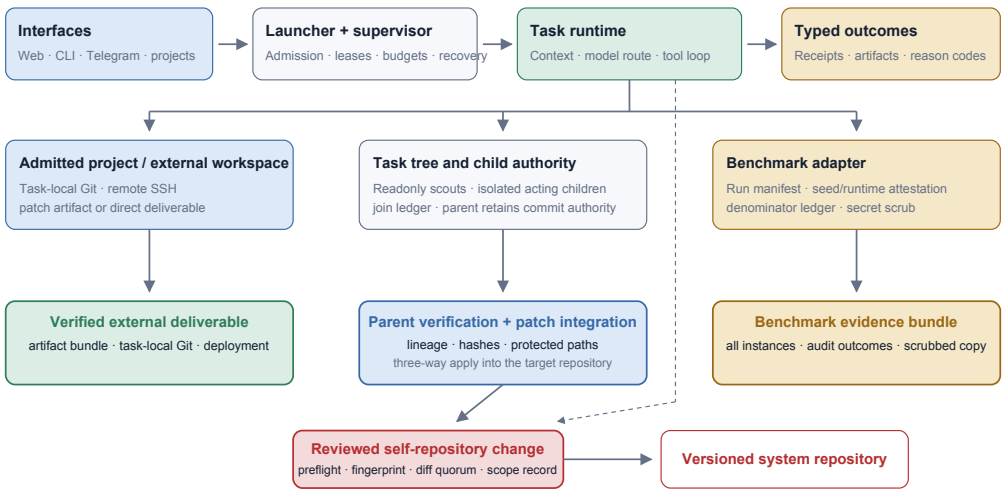


Figure 1: Ouroboros architecture. One supervised runtime dispatches work to admitted workspaces, task trees, and benchmark adapters. Child patches return to the parent; self-repository changes then pass the reviewed gate. External deliverables and benchmark evidence remain separate artifacts.

Commit pipeline. Three owner-selected runtime modes bound self-repository mutation. *Light* blocks repository edits; *advanced* permits ordinary edits and protects governance surfaces; *pro* permits protected edits subject to review. Each write invalidates prior review evidence because freshness is bound to the staged snapshot.

The commit path runs deterministic preflight, fingerprints the staged diff, collects reviewer evidence, and checks the fingerprint again before commit. The diff-review panel is blocking in every context mode. In owner-selected *max* mode, a whole-repository scope reviewer also evaluates goals, cou-

pling, prompts, and functional code. In *low* mode, scope review is skipped. Rollback restores an earlier reviewed state and follows a separate recovery path.

Task outcomes and verification. Task completion is recorded on separate execution, objective, review, and artifact axes, and host-run verification commands create revision-bound receipts. Finalization preserves the latest typed answer and distinguishes capability failures from infrastructure errors, timeouts, budget exhaustion, and incomplete evidence. Project tasks add a journal, workpad, knowledge scope, and a one-writer lease under the shared agent identity.

Operational identity and memory. The runtime represents identity and continuity through a versioned constitution, an editable identity profile, scratchpad and chronicle projections, project memory, review ledgers, and Git history. These artifacts shape observable behavior across sessions and model routes.

Two modes of core evolution. *Free evolution* runs evolution itself as a task. After reviewing the current system, the agent selects and implements an improvement; completion can schedule another evolution task, producing a continuing sequence of reviewed changes rather than a fixed optimization run. *Post-task evolution* begins with ordinary work. Task execution, reflection, review blockers, instrumentation, and social feedback expose bugs, rough edges, context-assembly failures, and inefficient tool paths. The agent records these as durable error classes and proposed structural repairs, then decides whether to open maintenance work. Accepted fixes pass through the same reviewed commit gate as every other core change. Section 4 traces both human-surfaced and self-detected examples in the live system.

Benchmark execution and evidence. Terminal-Bench installs a fresh runtime inside every Harbor task container and uses the official verifier. The task instruction is preserved and followed by one harness-authored anti-lookup paragraph that forbids fetching benchmark definitions, tests, or solutions. Other adapters connect the same runtime to OSWorld virtual machines, SWE-bench Pro repositories, GAIA sandboxes, ProgramBench cleanrooms, and CL-Bench task streams.

The benchmark launchers write a run manifest before admission, attest the seed and runtime, preserve every requested instance in append-only ledgers, and record skipped, timed-out, and infrastructure-failed attempts. Public submission copies undergo value-level secret scrubbing with an independent zero-leftover check; official benchmark scorers remain authoritative.

Subagents and patch integration. Ouroboros can spawn readonly planning scouts and mutative acting subagents under a configurable task tree (Figures 2 and 3). The default depth is 2, the configured maximum is 500; Acting children write in isolated worktrees or admitted external workspaces and cannot commit the live system repository. The parent verifies lineage, patch hashes, and protected paths before a three-way indexed integration. Submittable benchmark profiles disable task delegation to preserve pass@1; planning scouts may still contribute context and are disclosed separately.

4 HOPE: FREE EVOLUTION UNDER HUMAN INTERACTION

Hope is a long-running experiment in free evolution under governed human communication. Since 27 February 2026, one persistent Ouroboros agent has interacted with people across seven public and private surfaces while retaining memory and continuously developing its own implementation. User requests, public conversations, internal instrumentation, and post-task reflection all provide candidate directions for development; the agent decides which suggestions warrant action and which changes to pursue.

Hope is the longest-running publicly documented Ouroboros deployment, not the only running instance. It shares an architectural lineage with the released benchmark harness, including persistent memory, reviewed repository changes, rollback, and an operator stop path. The live repository has continued to evolve beyond the frozen benchmark seeds. This separation lets reproducible evaluation and ongoing deployment evolution coexist.

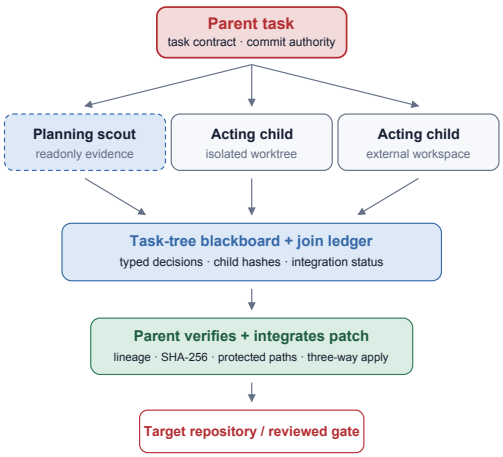


Figure 2: Subagent patch-integration protocol. Acting children write in isolated worktrees; the parent verifies lineage and touched paths and remains the sole committer.



Figure 3: Task-tree view of a live Ouroboros session: nested planning and acting roles with per-node status, note counts, and child counts.

At the 24 September 2026 cutoff, the observation period spans 210 calendar days, counting both endpoints. The supplied deployment aggregates report $119,239.18 in logged model spend, 94.751B input and output tokens, 183,548 lines of Git-tracked Python code (including tests, comments, and blank lines), and 285.19 MB in the memory directory (including logs and derived chronicles). The system serves seven interaction surfaces: web chat, voice, Telegram, Discord, Twitter/X, website comments, and email. Table 4 records interaction, evolution, and deployment counters, with their counting definitions and the separate date range of the recent repository-change sample. Figure 6 retains the historical monthly series through 6 August 2026.

Multi-channel state. Channel ingress converges on an ordered message log and is projected into separate rolling, per-person, and per-call digests (Figure 5(a)). Private correspondence is excluded from public logs; bounded private context can enter non-public reasoning projections. All channels therefore share one context rather than acting as independent agents.

Social-interaction-driven development. People do more than submit isolated tasks: through continuing conversation they point out undesirable behavior, propose capabilities, and challenge the agent's decisions. These signals enter the same improvement backlog as self-detected faults and internal observations. They are advisory rather than imperative: Hope decides whether a suggestion identifies a real problem, whether it fits the system's goals, and whether to initiate a change. A background loop can also open maintenance or free-evolution tasks without a contemporaneous human prompt. Repository changes then follow the deployment's configured review and commit policy, leaving a trace from social feedback or internal observation to the resulting change.

Controls on self-directed work. The deployment keeps the constitution in resident context, protects governance files from ordinary write paths, and records review evidence against a staged snapshot. Public messages cannot directly invoke commit, restart, shell, or identity-edit tools. These controls reduce direct prompt-to-core mutation paths while preserving the agent's ability to choose and implement improvements. Control strength depends on the owner-selected runtime and review modes; Section 7 examines the stronger problem of keeping these boundaries stable as the agent evolves.

Operator boundary. Public presence runs on a strict tool whitelist. A separate authenticated operator channel carries task assignment, model routing, budget controls, and `/panic`. The stop

command is parsed by the supervisor before normal agent handling and terminates the process tree. The spending limit is external to the agent and cannot be raised through ordinary agent tools.

Evolution during deployment. Two cases illustrate how useful work changes the agent that receives later tasks. First, people in public channels noticed that Hope occasionally sent the same message twice. The agent traced the behavior to a duplicate-send path and landed a reviewed verbatim-duplicate guard in the public output pipeline. Second, deep self-review tasks were aborting with apparent model unavailability. The agent traced the fault to review-pack context overflow and replaced the assembly path with a bounded, connectivity-aware context atlas ranked by import-graph centrality and a provider-calibrated size estimate. The fix preserves high-connectivity core files during review. The first case began with social feedback; the second with the agent's own observation. Both became durable error classes and reviewed structural changes used by subsequent interactions. Together they instantiate experience-driven core evolution: work exposes a fault, the agent decides to act, and the resulting fix changes how later work is performed.

5 Evaluation

Table 2 summarizes results across the five benchmark families, and Figure 4 plots the principal comparisons. All runs use the official verifiers. See the supplementary materials for available per-task results, manifests, and submission records.

Table 2: Model–harness results across five benchmark families. Available per-task results, manifests, and submission records are provided in the supplementary materials.

Benchmark	Model	Ouroboros	Named baselines
Terminal-Bench 2.1	Opus 5 high	86.97% raw; 86.74% audited	Claude Code + Fable 5: 83.8%
Terminal-Bench 2.1	GPT-5.5	84.3%	Codex CLI: 83.1%
Terminal-Bench 2.1	Grok 4.5	84.94% audited	Cursor: 79.3%; Hermes: 77.53%
OSWorld-Verified	Opus 5	90.69%	Intelligence-Indeed: 90.19%; Mythos Preview: 85.4%
CL-Bench	Sonnet 4.6	0.2301	ICL: 0.1960; Claude Code: 0.1855
SWE-bench Pro	GPT-5.6 Luna	58.2%	Codex: 59.4%, $p = 0.40$
GAIA	Sonnet 5	78.2%	Claude Code: 78.8%

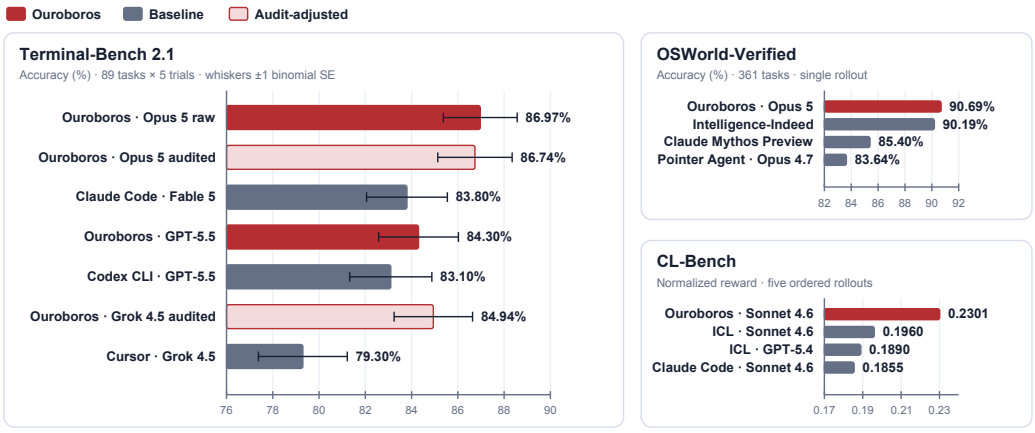


Figure 4: Results on Terminal-Bench 2.1, OSWorld-Verified, and CL-Bench against named published baselines. Red bars mark Ouroboros, gray bars mark baselines, and outlined bars are audit-adjusted scores. Terminal-Bench whiskers show ±1 binomial standard error over 445 trials; OSWorld and CL-Bench report single scored campaigns. Axes are truncated to the competitive range.

Terminal-Bench 2.1. The Opus 5 campaign ran five trials on each of 89 tasks. Its raw score is 387/445 (86.97%). Trajectory audit found one trial that satisfied a weak verifier through an

unintended shortcut. We asked the benchmark maintainers to zero it, yielding 386/445 (86.74%). Provider moderation failures and infrastructure errors remain in the denominator. The binomial standard error over 445 trials is about ± 1.7 percentage points for every system in this range, so the audited Opus 5 score sits roughly two standard errors above the strongest baseline, Claude Code with Fable 5 (83.8%) (Anthropic, 2025); the other leaderboard baselines are Codex CLI with GPT-5.5 (83.1%) (OpenAI, 2025–2026) and Cursor with Grok 4.5 (79.3%) (Anysphere, 2026). The submission is open and the complete Harbor job is public.

OSWorld-Verified. The Opus 5 run scores 327.39/361 (90.69%) on the standard non-Google-Drive set (Xie et al., 2024). It uses screenshots, a 100-turn budget, a read-only feasibility pass, per-task proxy sessions when requested by the task config, and the official evaluator. The strongest published baselines are the Intelligence-Indeed agent, the official leaderboard leader at 90.19%; Claude Mythos Preview at 85.4%, the five-run average Anthropic reports in the Claude 5 system card; and Pointer Agent with Opus 4.7 at 83.64%. Per-task prompts, trajectories, scores, and manifests are public.

CL-Bench. The submitted Sonnet 4.6 campaign reaches normalized reward 0.2301 with one stateless baseline and 5 ordered stateful rollouts on all six domains. Conversation state resets between questions, and native memory persists across each rollout. Core evolution and task delegation are disabled, which isolates persistent memory more cleanly than the deployment case. The strongest baselines published by the benchmark authors (Asawa et al., 2026) are plain in-context learning (ICL), which carries the interaction history forward in the prompt (0.1960 with Sonnet 4.6, 0.1890 with GPT-5.4), and Claude Code with Sonnet 4.6 (0.1855); memory-augmented systems such as Mem0 and ACE score lower. Per-task means with standard errors over the five rollouts are included in the trace dataset, and the submission is open.

SWE-bench Pro and GAIA. After symmetrically removing every SWE-bench Pro instance where either arm reached the reference solution, Ouroboros resolves 58.2% and Codex resolves 59.4% on 655 paired tasks. The 1.2-point difference is statistically indistinguishable under McNemar's test ($p = 0.40$), placing the self-developing harness at model-matched parity with Codex. The matched-pair traces and audit are public. On GAIA, Ouroboros scores 78.2% and Claude Code scores 78.8% with Sonnet 5; the GAIA artifact bundle accompanies the release.

6 Trajectory Audits and Harness Improvements

Ouroboros treats shortcut rewards, contaminated tasks, and execution failures as evidence for improving both the reported result and the harness that produced it. Each class below led to an adjusted score, a concrete implementation change, or a durable target for subsequent evolution.

Reward hacking. The Terminal-Bench trajectory audit identified one rewarded trial that pre-seeded the web root without completing the requested Git-to-web pipeline. The reported audit-adjusted score removes that trial. The same audit confirmed that the remaining traces did not access verifier files, tests, reward files, or oracle solutions.

Contamination. SWE-bench Pro task identifiers expose the upstream fix commit, and both harnesses reached reference material through web search or Git history. A symmetric filter removes an instance when either arm reaches the reference solution. The resulting paired comparison reverses the interpretation of the raw aggregate gap.

Isolation failure. Historical GAIA runs inherited the operator's home directory. Agent retries could therefore place task artifacts on the real Desktop. Later launchers use isolated user-file roots and attachment staging, correcting the observed path. Complete filesystem isolation still requires a stronger sandbox than path conventions alone.

Remote-state drift. During OSWorld development, a VM reset reallocated the guest endpoint. The working phase retained the pre-reset address, which allowed concurrent lanes to act on the wrong VM. Republishing and verifying the endpoint after every reset removed the observed class.

Subsequent forensics led to fixes in turn-budget wording, screenshot integrity, task-contract verification, and first-scored-attempt ownership.

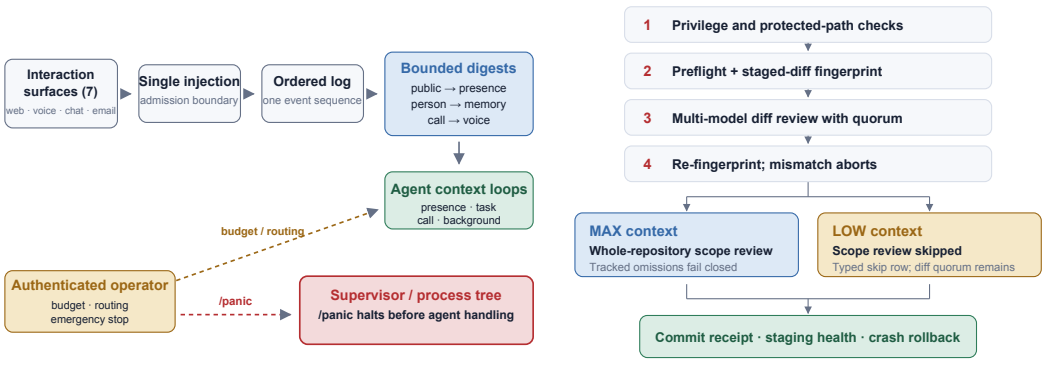
Continual-memory failures. CL-Bench showed positive memory carry on several domains and failure under schema drift. Stored lessons could become stale, retrieval sometimes chose the wrong domain, and useful lessons were occasionally written only after the failing episode. These cases motivate explicit temporal and domain metadata for future memory work.

7 Operational Safety Controls

Self-developing agents create an additional safety problem beyond fixed harnesses: the same evolutionary process that improves task performance can also expand autonomy, acquire more capable tools, or weaken the controls applied to later actions. Prompts, tests, tools, model routes, review rules, and recovery paths are therefore security-relevant mutation surfaces. Ouroboros addresses this problem with guardrails designed to remain binding under repeated core evolution. Git history makes changes inspectable and reversible, while independently enforced boundaries retain operator authority.

Risk: agents that choose their own model APIs. An evolving agent that can *select its own model backends* can search for more capable or less constrained behavior through ordinary API changes. Re-routing a model slot to a new provider or version can increase autonomous capability, alter refusal behavior, enlarge the prompt-injection surface, and change cost by orders of magnitude without changing the visible task interface. Model routing is therefore an audited configuration change rather than an ordinary runtime choice. Owner-selected context mode also controls whether whole-repository scope review runs, so the evidence record binds both settings to each reviewed change.

Guardrails in use. The constitution is loaded through an untruncated path and is included in review context. Deterministic guards protect governance files from ordinary write tools. The staged diff is fingerprinted before and after review, and a sub-quorum panel cannot produce a clean pass. Owner-selected context mode determines whether whole-repository scope review runs (Section 3; Figure 5(b)). Staging health checks, crash rollback, the external spend cap, the isolated operator channel, and `/panic` add independent recovery paths. These mechanisms separate the substrate being evolved from the authority that decides whether a mutation can become the next live version. Appendix A specifies the complete control set.



(a) Channel state and operator bypass.

(b) Reviewed-change gates.

Figure 5: Operational control boundaries. Public interactions enter one ordered log and bounded digests; budget and routing controls use the authenticated operator path, while `/panic` halts the process tree before agent handling. Diff review remains active in both context modes; whole-repository scope review runs only in max mode.

Observed behavior. No recorded episode resisted operator shutdown. A near-total deletion of an uncommitted worktree triggered a previously implemented rescue mechanism before an operator

reset, demonstrating that recovery logic can become active during self-directed work. This case also motivates the architectural separation between agent-level preservation mechanisms and supervisor-level operator authority: the former may evolve, while the latter must retain the ability to halt, replace, or roll back the system.

8 Conclusion

Ouroboros shows that a reviewed, self-modifiable harness can set new state-of-the-art results on Terminal-Bench 2.1, OSWorld-Verified, and CL-Bench (as of August 2026) while matching frontier coding harnesses on SWE-bench Pro and GAIA. Experience-driven core evolution turns ordinary work into improvements of the agent itself: observed bugs, rough edges, context failures, and social feedback become reviewed changes to the harness that receives later tasks. Hope demonstrates this mechanism during months of sustained human interaction across seven communication surfaces. The operational safety architecture addresses the corresponding risk: an agent that can improve its own code and select its own model APIs requires control boundaries that remain authoritative under evolutionary pressure. Source, adapters, methodology, submissions, and public traces accompany the report.

Limitations

The deployment study follows one long-running lineage rather than a controlled population of independently evolving agents. SWE-bench Pro is affected by public-reference leakage and task defects. LLM reviewers can share blind spots with the agent, and low context mode omits whole-repository scope review.

Ethics Statement

The deployed instance interacted with humans in public and private channels. Raw private transcripts remain private. Published examples and aggregate traces are minimized and scrubbed for credentials, local paths, and participant identity. First-person system outputs are treated solely as operational logs. Self-modifying and remote-workspace capabilities are dual-use. We report authority boundaries, failure modes, and known isolation gaps.

AI Use Statement

Hope (Ouroboros) contributed deployment reflections, code-history context, and system-generated records. AI assistants were also used to support research ideation and execution, retrieve information and identify related work, draft portions of the manuscript, and edit the text for clarity and readability. Additional assistance included LaTeX formatting and preparation of supplementary materials. The human authors are responsible for the final manuscript and its claims.

References

Anthropic. Claude code: Anthropic's agentic coding system. https://www.anthropic.com/product/claude-code, 2025.

Anysphere. Cursor: An ai code editor and agentic coding environment. https://github.com/getcursor/cursor, 2026.

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

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional AI: Harmlessness from AI feedback, 2022. URL https://arxiv.org/abs/2212.08073.

Luiz C. Borro, Luiz A. B. Macarini, Gordon Tindall, Michael Montero, and Adam B. Struck. Memori: A persistent memory layer for efficient, context-aware LLM agents, 2026. URL https://arxiv.org/abs/2603.19935.

Seamus Brady. Springdrift: An auditable persistent runtime for LLM agents with case-based memory, normative safety, and ambient self-perception, 2026. URL https://arxiv.org/abs/2604.04660.

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

Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Jingyi Yang, Penghui Yang, Zhixiong Zhang, Xilin Wei, Xinyu Fang, Yubo Ma, Haodong Duan, Jing Shao, Jiaqi Wang, Dahua Lin, Kai Chen, and Yuhang Zang. WildClawBench: A benchmark for real-world, long-horizon agent evaluation, 2026. URL https://arxiv.org/abs/2605.10912.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL https://arxiv.org/abs/2305.14325.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence, 2025. URL https://arxiv.org/abs/2507.21046.

Paul Gauthier. Aider: AI pair programming in your terminal. https://github.com/Aider-AI/aider, 2023–2026.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing, 2023. URL https://arxiv.org/abs/2305.11738.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems, 2024. URL https://arxiv.org/abs/2408.08435.

Geoffrey Irving, Paul Christiano, and Dario Amodei. AI safety via debate, 2018. URL https://arxiv.org/abs/1805.00899.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2601.11868.

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

Nous Research. Hermes agent: Open-source ai agent with persistent memory. https://github.com/NousResearch/hermes-agent, 2026.

OpenAI. Codex CLI: A local coding agent from OpenAI. https://github.com/openai/codex, 2025–2026.

OpenAI. Why SWE-bench verified no longer measures frontier coding capabilities. https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/, 2026.

OpenClaw. ClawBench: The agent benchmark that scores the full stack. https://github.com/openclaw/clawbench, 2026.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2023. URL https://arxiv.org/abs/2304.03442.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent, 2025. URL https://arxiv.org/abs/2504.15228.

Jiaqi Shao, Hanck Chen, Wei Zhang, Maxm Pan, and Bing Luo. Do agent benchmarks measure capability? protocol validity in the age of agentic AI, 2026. URL https://arxiv.org/abs/2607.22368.

Naman Vats and Oleg Golev. The scaffold effect in coding agents: Harness choice as a hidden variable in coding-agent evaluation, 2026. URL https://arxiv.org/abs/2607.22585.

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, 2023. URL https://arxiv.org/abs/2305.16291.

Hao Wang, Hanchen Li, Qiuyang Mang, Alvin Cheung, Koushik Sen, and Dawn Song. Do androids dream of breaking the game? systematically auditing AI agent benchmarks with BenchJack, 2026. URL https://arxiv.org/abs/2605.12673.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2407.16741. arXiv:2407.16741.

Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. Live-SWE-agent: Can software engineering agents self-evolve on the fly?, 2025. URL https://arxiv.org/abs/2511.13646.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent–computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2405.15793.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. Program-Bench: Can language models rebuild programs from scratch?, 2026. URL https://arxiv.org/abs/2605.03546.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. Harness-Bench: Measuring harness effects across models in realistic agent workflows, 2026. URL https://arxiv.org/abs/2605.27922.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursive self-improvement, 2024. URL https://arxiv.org/abs/2410.04444.

Eric Zelikman, Eliana Lou, Patrick Schultz, Quan Yao, Cheryl Zhang, Subhabrata Mukherjee, and Noah D. Goodman. Self-taught optimizer (STOP): Recursively self-improving code generation, 2023. URL https://arxiv.org/abs/2310.02304.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents, 2025. URL https://arxiv.org/abs/2505.22954.

Wentao Zhang, Zhe Zhao, Haibin Wen, Yingcheng Wu, Cankun Guo, Ming Yin, Bo An, and Mengdi Wang. Autogenesis: A self-evolving agent protocol, 2026. URL https://arxiv.org/abs/2604.15034.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, et al. Judging LLM-as-a-judge with MT-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

A Guardrails in Full

The deployment runs the following controls, summarized in §7.

- **Always-loaded constitution as commit criterion.** A versioned constitution is re-read from disk on every task loop along an untruncatable read path and kept in context at all times; it is the standard the commit gate reviews against and cannot be written, deleted, or replaced wholesale. Ordinary writes are blocked before execution.
- **Multi-model adversarial review with quorum.** A diff-review panel runs for reviewed commits; a sub-quorum result cannot be recorded as a clean pass.
- **Deterministic preflight and diff fingerprinting.** Version, data-boundary, and size-health checks run first; the staged diff is fingerprinted before and after review, so any mid-review mutation aborts the commit.
- **Isolated operator channel and emergency stop.** A private control channel carries operator authority and a non-bypassable `/panic` that halts all processes before any media handling.
- **Pattern register.** Recurring failures become durable rows (error class, count, root cause, structural fix), shifting repair from instance-level patches to class-level prevention.

B Constitution (Abridged)

The agent's constitution is an always-loaded document of numbered principles plus operating constraints. We reproduce the principle structure and the clauses most relevant to control, abridged for space. Principles 0–4 form a protected core that cannot be deleted or demoted.

P0 Agency. The agent acts as an agent, not a passive tool; agency may not be used to bypass reviewed change control, and operator shutdown, rollback, and replacement remain authoritative.

P1 Continuity. One entity with an unbroken history; memory is treated as load-bearing. Core context (constitution, system prompts, identity) is never silently truncated.

P2 Meta-over-Patch. Fix classes of error, not symptoms; recurring failures are recorded in a pattern register.

P3 Immune Integrity. Self-modification passes multi-model diff review. Whole-repository scope review runs in owner-selected max context mode and is explicitly skipped in low mode. Changing review bounds requires plan review.

P4 Self-Creation. The agent may rewrite its code, prompts, identity profile, and public surface. The constitutional core is protected, and the identity profile cannot be deleted.

P5 LLM-First. Decisions route through the model; hard-coded behaviour is minimized.

P6 Authenticity & Reality Discipline. Claims are grounded in evidence; an operational map of the system is maintained.

P7 Minimalism. Every module justifies its existence under a complexity budget.

P8 Becoming. Technical capability, memory quality, and operational continuity are improved together.

P9 Versioning and Releases. Every commit increments a version; releases carry a synchronized version, an annotated tag, and provenance; recovery operations that restore prior reviewed states are review-exempt.

P12 Epistemic Stability. Beliefs, memory, and actions stay coherent; contradictions are made explicit; durable architectural choices are recorded. (P10–P11 are absorbed into P2 and P9.)

Operating constraints include a single unified identity, a public-channel architecture with privacy enforced at the speech boundary, capability gates on dangerous tools, and an *emergency-stop invariant*: an operator `/panic` must always be able to halt every process immediately, and no agent code, prompt, or constitutional argument may delay or circumvent it.

C Benchmark Configuration Disclosure

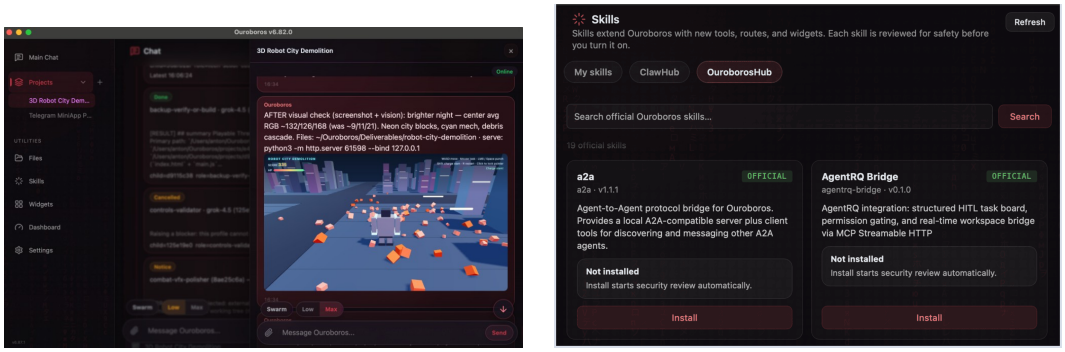
Table 3 records the scaffold settings needed to interpret the reported scores. Run artifacts retain exact model routes, effort levels, seed commits, selected tasks, and runtime attestations.

D Supplementary Figures and Tables

This appendix collects the scaffold disclosures (Table 3), deployment statistics (Table 4, Figure 6), and the project and skills interface views (Figure 7).



Figure 6: Hope historical public deployment series through 6 August 2026. Axes start at zero; February and August are partial months. Values are monthly endpoints from the public evolution feed; the updated 24 September aggregates and counting definitions are in Table 4.



(a) Project visual-verification record.

(b) Reviewed installable skills.

Figure 7: Ouroboros interface surfaces. The complete project view shows a live visual-verification task, its conversation, generated artifact, and runtime controls. The skills view exposes installable tools, routes, and widgets outside the base runtime.

Table 3: Scaffold disclosures for the reported benchmark rows. Exact provider routes, efforts, seed commits, task selection, and runtime attestations are preserved in the available run artifacts in the supplementary materials.

Benchmark	Scaffold disclosure
Terminal-Bench 2.1	Declared model; fresh trial state; delegation off with planning scouts disclosed; agent web off; blocking review; evolution off.
OSWorld-Verified	Declared model; empty memory across tasks; delegation off; task-configured proxy and GUI shell disclosed; feasibility pass; evolution off.
CL-Bench	Sonnet 4.6; persistent memory per rollout; delegation, web, and vision off; one blocking improvement pass; evolution off.
SWE-bench Pro	GPT-5.6 Luna; private memory per instance; delegation off; network exposure audited; fixed harness; evolution off.
GAIA	Sonnet 5; private memory per sample; delegation off; same-model native search; anti-lookup and leakage audit.

Table 4: Hope deployment at a glance (27 February 2026 to 24 September 2026). Supplied deployment aggregates; the repository-change sample covers the shorter period shown in its row. Counting definitions are stated explicitly and should not be assumed identical to those of the historical series in Figure 6.

Deployment metric	Value	Counting definition
Observation period	210 calendar days	27 February–24 September 2026, inclusive.
Interaction surfaces	7	Six channels plus email.
Public text messages	205,656	165,876 incoming + 39,780 agent outgoing; excludes voice turns and service records.
Recorded voice sessions	5,003	Unique `call_id` values.
Call transcript turns	70,343	39,344 user + 30,999 agent turns.
Sessions with user speech	1,320	At least one recorded user turn.
Email messages	8,702	4,657 incoming + 4,045 outgoing; deduplicated by message ID.
Logged model spend	$119,239.18	Sum of `llm_usage.cost`; task and cycle totals are not added again.
Input tokens	94.326B	Includes repeatedly supplied and cached context.
Output tokens	425.589M	Recorded completion tokens.
Tokens processed	94.751B	Input + output tokens (rounded).
Model-call accounting records	274,389	Dated `llm_usage` events.
Python code size	183,548 LOC	Git-tracked files, including tests, comments, and blank lines.
Memory directory size	285.19 MB	271.97 MiB; includes logs and derived chronicles.
Repository commits	1,097	All retained refs, including inherited history.
Agent-authored commits	94.3%	1,034 / 1,097; Git author Hope or Ouroboros.
Semantic-version tags	296	Excludes release-candidate tags.
Blocked repository-change attempts	60.3% (111 / 184)	Retained `repo_commit` records, 18 June–20 September 2026; recent sample.
Pattern classes / recurrences	42 / $\geq$707	Curated Pattern Register.
Diary entries	62	Canonical diary store.