Prime Agent: A Self-Improving RLM Harness

Seth Karten[♠♡♢] Alex L. Zhang[♡♣♢] Kevin Thomas[♡] Sebastian Müller[♡]
Elie Bakouch[♡] Daniel Auras[♡] Mika Senghaas[♡] Fares Obeid[♡]
Konstantin Dunas[♡] Johannes Hagemann[♡] Sami Jaghouar[♡]

[♠] Princeton University [♡] Prime Intellect [♣] MIT
[♢] Correspondence: seth@primeintellect.ai, altzhang@mit.edu

First published: August 5, 2026 • Current version: August 24, 2026

Abstract

Language models are sequential processors, but long-horizon agency requires external information and computation beyond model weights and active context. Prime Agent is an open-source harness for long-horizon evaluation and coding-agent workflows. A persistent IPython REPL follows the Recursive Language Model abstraction for programmatic context processing and test-time compute, while Continual Harness preserves histories, memories, skills, prompts, and subagent specifications across trajectories. Recursive subagents coordinate through direct agent-to-agent communication, and the Agents View lets humans inspect and manage daemon-backed sessions. Prime Agent standardizes execution, recovery, verification, and resource accounting while leaving strategy construction to the model. This low-friction, expressive membrane prevents harness failures from becoming model failures and pushes measurement toward the model's true maximal underlying capability. Prime Agent raises ARC-AGI-3 RHAE Best@1 from 30% to 95.5% and matches or exceeds native and popular harnesses across long-context coding, GPU-kernel generation, emulator construction, and autonomous nanoGPT speedruns. On Factorio, we find refinement allows for continuous technology progression and dedicated subagents enable parallelized work. Code is available at https://github.com/PrimeIntellect-ai/prime-agent.

1 Introduction

A strong language model on its own does not have the full capabilities of a computer. An LLM is a bounded sequential processor whose next decision can use only state information exposed in its weights and active context. A harness supplies the missing computational

substrate that allows for external actions via tool-calls. These external actions also include information management beyond prior knowledge stored in the model weights. Context management was first enabled by agentic compaction, the process by which a model selectively analyzes its own context to reduce tokens while keeping essential information. However, the full information state has grown beyond the weights and token context. Imagined as a state information cache (Figure 2), model weights are L0, active context is L1, a persistent REPL and recursive subagents form L2, and disk-backed history, memories, and skills form L3. This makes the system more *von Neumann-like*: the model can read, transform, and write addressable state outside the instruction currently being generated [34, 35].

This perspective makes *expressivity* the key property of a harness. Rather than encode one workflow, an expressive harness exposes primitives from which the model constructs programs, subagents, and feedback loops at inference time. Recursive Language Models (RLMs) make context and recursive invocation programmable [44]; Continual Harness makes prompts, subagents, skills, and memories revisable from the trajectory history [19]. Lastly, we enable large-scale coordination and orchestration of multi-agent swarms through direct agent-to-agent communication. These components let a fixed model use information management and test-time compute to expand its reachable strategy set.

Prime Agent is designed first as a standardized harness for *long-horizon evaluation*. The most popular metric of which is score at a fixed expenditure [8], such as cost/tokens and time, or, especially for long horizon, score at practical plateau [8], which allows us to analyze the shape of performance over time. The harness is the membrane through which the model observes and acts on the world. A model should fail an evaluation because the task exceeds its capability, not because the harness dropped state, restricted useful actions, miscounted resources, or terminated prematurely. Prime Agent therefore combines standardized, reliable execution with a low-friction and expressive interface for programmatic tools, information management, and swarm management. This lets the model fully use its test-time compute, pushing measured performance toward its true maximal underlying capability rather than the limitations of its harness.

Prime Agent jointly manages information and computation. Information management moves state across L1–L3 through programmatic context processing, compaction, persistent histories, and revisable memories. Computation management allocates test-time compute to programs, tool calls, reusable skills, and parallel recursive subagents [32, 44]. Direct agent-to-agent communication connects the two, routing information across distributed computation so the swarm can coordinate dynamically rather than follow a fixed graph. These communication links also let humans inspect, message, attach to, and intervene in subagent sessions without following every exchange [17, 21]. Together, these interactions produce retained trajectories that improve future computation and can train later model generations [25, 40, 43].

We present Prime Agent, an open-source harness for long-horizon model evaluation and coding-agent workflows. Prime Agent integrates information and computation management across active context, persistent programmatic execution, recursive subagents, and retained histories, memories, and reusable skills, connected through direct agent-to-agent and human-agent communication. Its Agents View provides a visual interface for inspecting, attaching to, and managing persistent daemon-backed agent sessions. We standardize evaluation infrastructure while preserving the model's freedom to construct its own strategy. Prime Agent improves ARC-AGI-3 performance from 30% to 95%, matches or exceeds Pi, Claude Code, and Codex (and outperforms other harnesses, Hermes Agent, OpenCode, and Kimi-Code, on other benchmarks) across long-context coding, GPU-kernel generation, and emulator

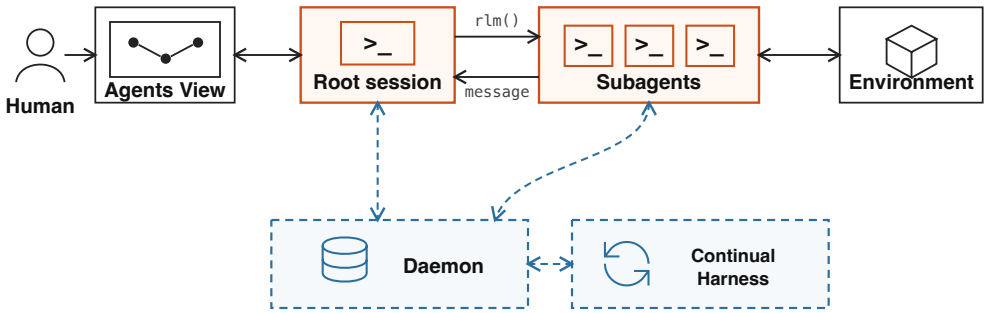


Figure 1: Prime Agent connects persistent root and subagent sessions to a daemon, Continual Harness, the Agents View, and the environment. Solid arrows carry execution and messages; dashed arrows carry persistent state.

construction, sustains an 85.5-hour nanoGPT run with 19 validated records, and supports four-character Factorio control and long-horizon MazeBench exploration.

2 Prime Agent Architecture

We next describe the Prime Agent architecture in detail. In particular, we discuss (1) how the system manages information and computation (§2.1), (2) how state is organized across model weights, active context, the REPL, and disk-backed storage (§2.2), (3) how persistent REPLs and RLM calls support programmatic computation (§2.3), (4) how recursive sessions communicate with agents and human operators (§2.4), (5) how Continual Harness turns trajectory evidence into reusable state (§2.5), and (6) how long-horizon controls define continuation, termination, and evaluation accounting (§2.6). Figure 1 provides an overview.

2.1 Architecture overview

Prime Agent separates information management from computation management. Information management determines what state enters a model invocation and what survives compaction or restart.

Computation management maps model-selected actions to code, tools, and recursive subagent sessions. Direct agent-to-agent communication connects related sessions, and direct human-agent interaction exposes individual nodes for inspection and intervention.

Models manipulate intermediate values with code. Sessions retain history across compaction, detachment, and restart. Subagents inherit the root's execution and communication primitives. The runtime records model calls, tool use, messages, harness changes, and resource use. The model controls decomposition, computation allocation, communication, and stopping.

2.2 Information hierarchy and persistent state

A model invocation is parameterized by fixed weights and conditions on its active token context. Prime Agent adds state outside that context. External state affects generation when the runtime injects it or an operation serializes a result into the context. Figure 2 organizes this state by visibility, access mechanism, and persistence.

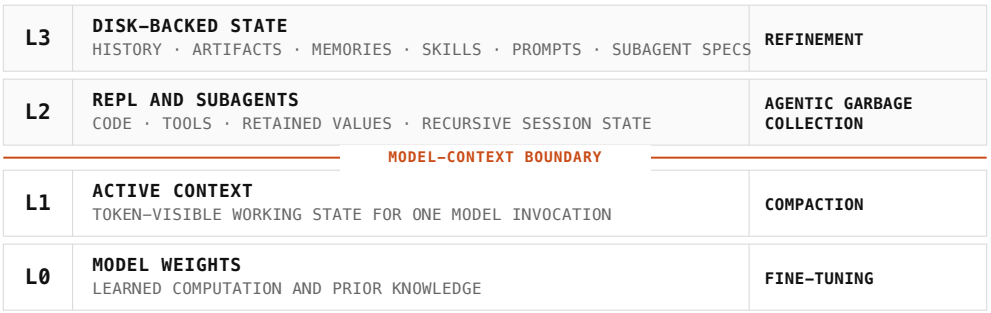


Figure 2: Prime Agent state hierarchy. The boundary between L1 and L2 separates token-visible model state from explicitly managed computation and retained state.

Each level changes through a different mechanism. Fine-tuning updates L0, compaction rewrites L1, and refinement versions selected L3 entries. We call the L2 mechanism *agentic garbage collection*. The model creates, retains, summarizes, or deletes REPL values and subagent sessions as the task changes.

Explicit operations move information between levels. Python values and tool outputs in L2 enter generation when serialized into L1. Compaction replaces a conversational prefix with a summary and retains the original events in L3 for REPL retrieval. The runtime assembles selected Continual Harness entries into later supplemental prompts; other L3 artifacts enter context on retrieval. L0 stays fixed.

The retained runtime state includes an append-only event history, selected kernel snapshots, the rooted session tree, context and compaction records, persistent message queues, and versioned Continual Harness state. Branching or forking creates a new logical continuation without deleting the prior event sequence. Recovery reconstructs the session under the same identity. Non-serializable Python objects and external processes are recreated from saved artifacts or external services.

2.3 Programmatic computation with RLMs

Each session owns a persistent IPython Read-Eval-Print Loop (REPL). At test time, compute comprises model inference, Python execution, and tool calls; evaluations report tokens, time, and cost separately. Installed tools are imported as Python modules for parsing, filtering, aggregation, and verification with ordinary code. Intermediate values persist across turns and remain outside active context until selected. This avoids repeatedly serializing large logs, task specifications, and structured evaluator output into the context.

Prime Agent implements the RLM abstraction with the asynchronous `rlm` primitive [44]. Calling `rlm` creates and schedules a subagent session, then returns a stable handle before the subagent completes. The subagent receives its own model context, IPython kernel, history, and workspace metadata. The parent continues local computation while subagents run. Results arrive later through direct agent-to-agent communication, and retained handles support follow-up after compaction or restart. The model chooses between local code, tools, sequential delegation, and parallel subagents. Prime Agent defines their execution semantics instead of a fixed workflow graph. A complete orchestration example appears in Section B.

2.4 Recursive orchestration and interaction

The daemon owns live sessions independently of the client that created them. Root and subagent sessions use the same lifecycle. Sessions are running during a turn or tool operation,

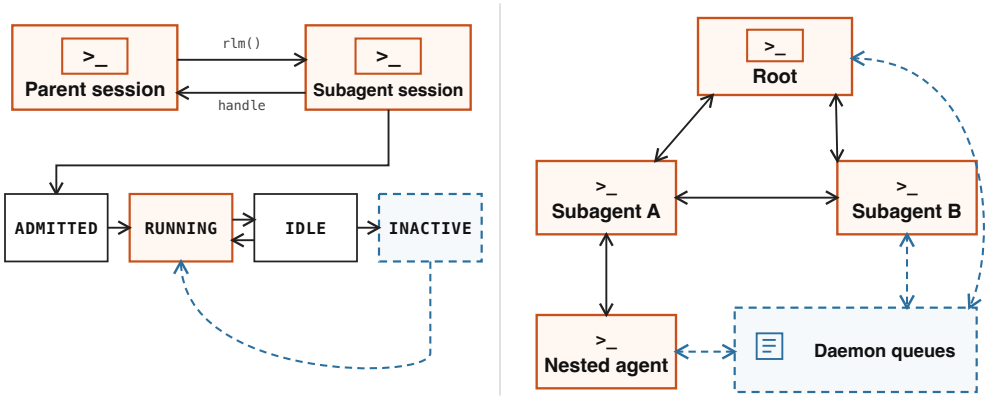


Figure 3: Multi-agent orchestration lifecycle and direct agent-to-agent communication.

idle when loaded without an active turn, and inactive when unloaded but recoverable from persistent state. Client detachment leaves the session running. Stable session and parent identifiers preserve the recursive topology across these transitions.

Direct agent-to-agent communication uses asynchronous, daemon-mediated queues. An agent addresses its parent, children, and siblings, and queued messages remain available when a recipient becomes active again. Filesystem, network, and credential access follow the permissions of the runtime environment. Figure 3 summarizes the shared lifecycle and communication topology.

The Agents View exposes the persistent tree for direct human-agent interaction. The interface lets a user inspect history, attach to a session, provide new input, or detach without interrupting execution. The `agent-observe` interface provides bounded read-only status and recent-message previews; `agent-message` targets a named related session. This allows for full interaction via the orchestrator.

2.5 Continual Harness

Continual Harness exposes supplemental state for trajectory-time reads and writes [19]. Prompt notes store behavioral instructions, memories store facts, skills package executable procedures, and subagent specifications store reusable roles or divisions of labor. Typed state separates rules, facts, programs, and coordination patterns. Entries support create, read, update, and delete operations; local entries belong to one session, and explicitly requested global entries remain available to later sessions.

Refinement converts trajectory evidence into versioned state updates. Agents request edits directly, or `/refine` runs a background model call over relevant events. The runtime applies each edit at a turn boundary, records its trigger and intended effect, and assembles supplemental state for the next invocation. Versions preserve provenance and enable rollback. Refinement supplements the immutable base prompt without rewriting foundational policy.

Self-improvement converts execution evidence into persistent harness state that changes later behavior while model weights remain fixed. Useful computations become skills, repeated coordination patterns become subagent specifications, and corrected assumptions become memories or prompt notes. The resulting trajectory record also provides training data for later model generations.

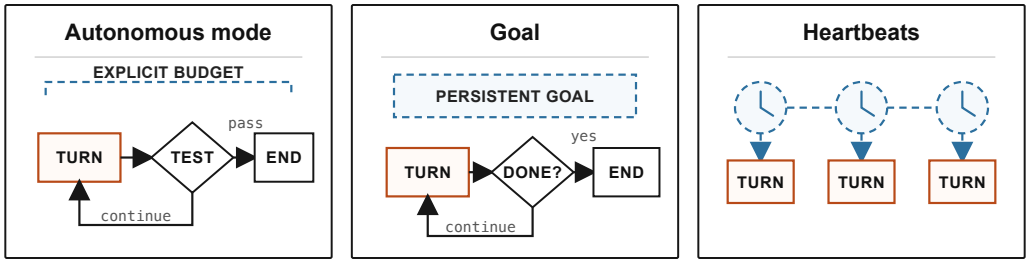


Figure 4: Long-horizon control mechanisms. Autonomous mode continues under an explicit budget until an end-condition test passes. Goals retain an objective across continuations until agentic completion. Heartbeats initiate cron or timed turns.

2.6 Long-horizon execution and evaluation semantics

Prime Agent exposes three long-horizon control mechanisms (Figure 4).

Autonomous mode continues model turns within an explicit budget and evaluates a task-specified end-condition test after each turn. A failed test returns bounded output for another attempt; turn, token, and wall-clock limits stop execution. A goal retains an objective across continuations and ends through agentic completion, when the agent marks the goal complete. Heartbeats initiate turns on cron or timed schedules.

Evaluation configurations bind task and tool interfaces to model and provider settings, compaction and refinement policies, retry policy, completion gates, and resource limits. Accounting aggregates the root and descendant sessions, so delegation remains visible in test-time cost. Event history links model and tool calls, messages, interventions, retries, verifier outcomes, and harness edits to that configuration. Standardized persistence, recovery, termination, and accounting separate harness failures from model failures while preserving model control over decomposition.

3 Evaluation

The evaluation addresses three research questions derived from Prime Agent's design. **RQ1: Test-time scaling.** Can a standardized, expressive execution interface let frontier models convert additional output tokens and API cost into verified task progress? We evaluate this question on ARC-AGI-3. **RQ2: Information management.** Can models use persistent REPL state to search, transform, and aggregate information across long contexts? We compare Prime Agent with native and alternative harnesses on long-context reasoning and coding tasks. **RQ3: Persistent recursive execution.** Can the same runtime sustain multi-day experimentation, iterative systems construction, recursive control, and online refinement? We study nanoGPT, PMPP-Hard, EmulatorBench, Factorio, and MazeBench through end-to-end outcomes and trajectory analysis. The trajectory analyses show how agents allocate subagents, retain information, and recover from disruption.

3.1 Interactive reasoning at test-time scale

ARC-AGI-3 is the clearest test of Prime Agent to support strong, consistent long-horizon evaluation [1]. Each game requires the model to learn the rules of the game, creating an ad-hoc world model under an action limit. Prime Agent supplies only the environment interface and an autonomous prompt adapted from PRO-LONG [9]; the model constructs the strategy. We note that Claude Code and Codex runs perform worse than Anthropic and

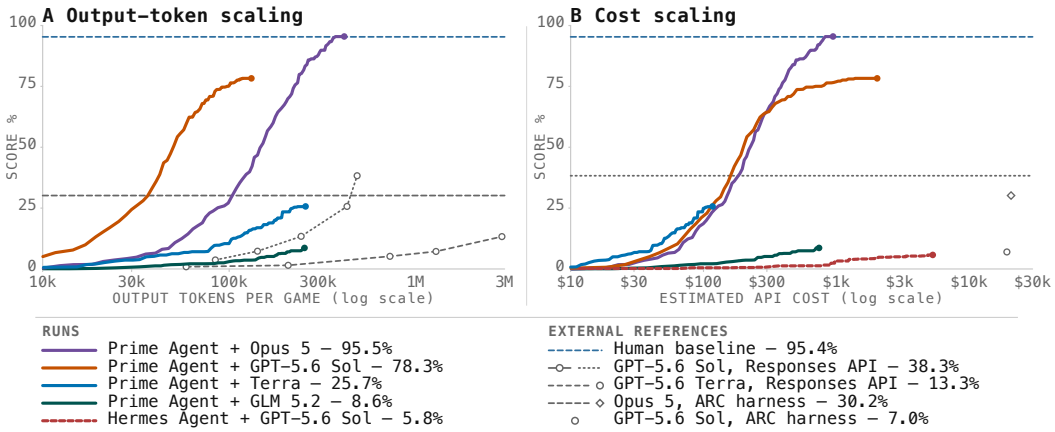


Figure 5: ARC-AGI-3 test-time scaling. RHAE score versus output tokens per game (left) and estimated API cost (right).

Open AI self-reported performance on ARC-AGI-3 (public set), so we defer to their results over our own runs with matched prompt and settings.

Across the observed configurations, additional output tokens and cost are converted into progress at sharply different rates (Figure 5). The stronger configurations continue to improve across a long interaction horizon, while others plateau early. This pattern is consistent with a model-controlled interface that permits model-dependent test-time scaling instead of imposing one fixed workflow. The reference lines and points place these curves beside official ARC results. They are external values because our native-harness reruns fell below the published scores, so they situate the result rather than isolate a causal harness effect.

3.2 Long-context information management

The long-context suite tests whether a model can actively manage information that does not fit naturally into one prompt. Prime Agent stores the initial context in a readable file, allowing the model to search, transform, summarize, and revisit it from the persistent REPL. This changes long-context reasoning from passive attention over a fixed sequence into a programmatic information-management problem. The suite covers aggregation, latent retrieval, instruction following, reasoning, and long-form coding [3, 5, 6, 23, 33].

		GLM-5.2 (reasoning: high)		Opus 5 (reasoning: high)		GPT-5.6 Sol (reasoning: high)	
Task	Setting	Prime	Pi-mono	Prime	Claude Code	Prime	Codex
OOLONG (Yahoo, 128k) [5]	long context	**.700**	.420	.900	**.920**	**.940**	.900
OOLONG-Pairs [44]	long output	**.874**	.556	**.929**	.922	**.911**	.895
OBLIQ-Bench (math) [33]	ranking (nDCG@10)	**.669**	.635	**.802**	.795	.612	**.646**
LongBench Pro (English) [6]	comprehension	**.777**	.768	**.804**	.790	**.794**	.790
LongBench v2 [3]	expert long tasks	.680	**.696**	.744	**.746**	**.714**	.704
ManyIH Coding [45]	long instructions	**.424**	.386	**.536**	.522	**.499**	.454
ManyIH IF [45]	long instructions	**.209**	.164	**.225**	.175	.216	**.232**
LongCoT-Mini [23]	long reasoning	**.638**	.613	**.722**	.558	.671	**.681**
EmulatorBench [18]	long coding	**.208**	.000	.047	**.062**	**.275**	.228

Table 1: Long-context results. Bold marks the higher point estimate within each nominal-model pair; metrics differ by row. Bold is not statistical significance, and uncertainty intervals are unavailable.

We generally find Prime Agent to be competitive across a wide range of long tasks, especially against the harness that did not use a model trained around it. Prime Agent especially excels at long-running or long-context tasks, and can competitively run on its own

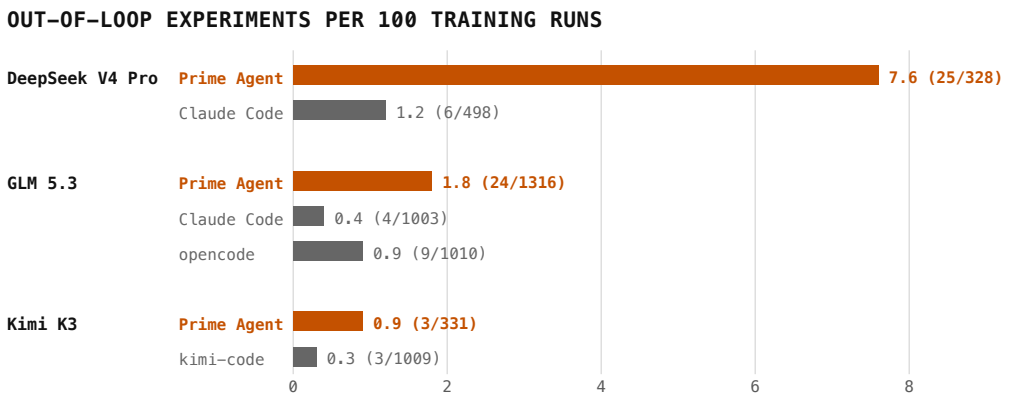


Figure 6: Out-of-loop experimentation across harnesses. Distinct experiments created outside the training script per 100 training-script executions, pooled over the 2–3 seeds per harness (labels: experiments/training runs). Counts are hand-classified from complete traces; denominators are audited where available and otherwise estimated from launch commands.

as an autonomous agent. We include a set of focused case studies and experiments on long settings where Prime Agent excels.

3.3 Multi-day autonomous research

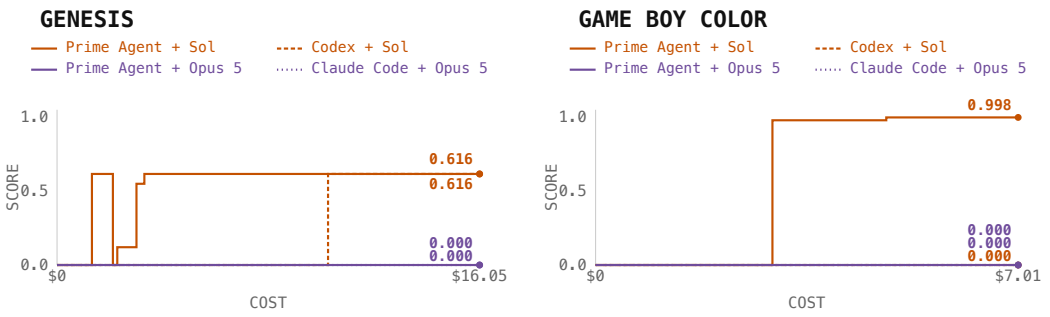
The nanoGPT speedrun [4] measures how far an agent can reduce the number of training steps required for a 124M-parameter GPT to reach a fixed validation loss, with each record verified as an eight-seed mean. For each of three models (Kimi K3, DeepSeek V4 Pro, and GLM 5.3), we compare Prime Agent against an alternative harness: the model developer's own CLI where one exists, and Claude Code or opencode otherwise. We find that the choice of harness has little effect on final records compared to the noise of the experiment.

Model behavior, however, differs. On Prime Agent, models regularly use the persistent REPL to experiment outside the benchmark's training script, for example by simulating a candidate optimizer on synthetic gradients or numerically optimizing update-rule coefficients before launching a training run. Figure 6 counts these experiments across 18 runs, normalized by the number of training runs each agent executed; Section A reproduces one such experiment per model. The effect is largest for DeepSeek V4 Pro, which created roughly six times more such experiments per training run under Prime Agent than under Claude Code. This may be due to the fact that DeepSeek's own agent harness provides a similar code-execution mode, so the REPL matches a workflow the model was likely trained on. We also observe that models construct programmatic interfaces to the benchmark itself: Kimi K3 defined a probe function through which it ran roughly ninety screening experiments and all 19 of its validated records, whereas the same model on its own CLI performed every operation through direct file edits and built no such machinery.

3.4 Programmatic systems construction

Emulators. An emulator is software that reproduces another computer system's observable behavior. We evaluate Prime Agent on EmulatorBench, a benchmark that tasks agents with constructing emulators in Rust for a variety of game systems. Agents are given a specification of the emulator and a set of diagnostic tests in the form of a verifier.

The correctness of an emulator is given from its ability to mimic the behavior of the target machine. This is measured by human-generated diagnostic programs that inspect



(a) Sega Genesis. **(b)** Game Boy Color.

Figure 7: Selected EmulatorBench runs. Stepwise verifier score versus estimated cost; hue encodes model and line style encodes harness.

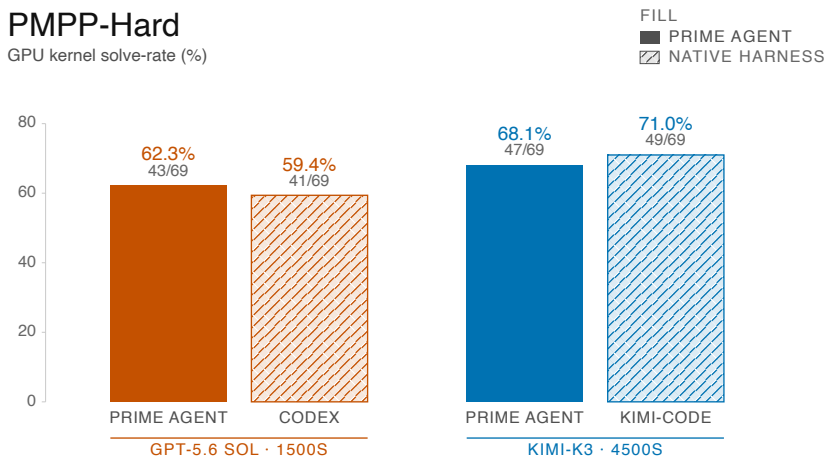


Figure 8: PMPP-Hard solve rates at fixed within-model budgets.

the emulator's behavior, such as the CPU flags, PPU timing, and other components. In an effort to minimize the effects of data contamination, we require the agent to build the emulator from scratch in Rust, sandboxed without any reference implementation. We report preliminary results in Table 1 on this long-context coding benchmark averaged over 16 emulator reconstructions, as well as two emulators in Figure 7, the SEGA Genesis and Nintendo Game Boy Color, that Prime Agent successfully reproduces. For Opus, our runs surprisingly failed to solve the tasks despite successful tool-call responses.

GPU kernels. PMPP-Hard compresses the same programmatic loop into repeated edit, compile, correctness-check, and profile cycles under a wall-clock budget.

Prime Agent and the native harnesses remain close, with the ordering reversing between the two model groups. In these reported within-model comparisons, the general persistent interface supports the compiler–profile loop with no large observed gap. One noted limitation of PMPP-Hard is the strict wall-clock budget comparison. What the wall-clock budgets do not reveal is the substantial improvement in token usage by models that use Prime Agent. This means that the same performance as Codex or Kimi-Code is achieved by Prime Agent

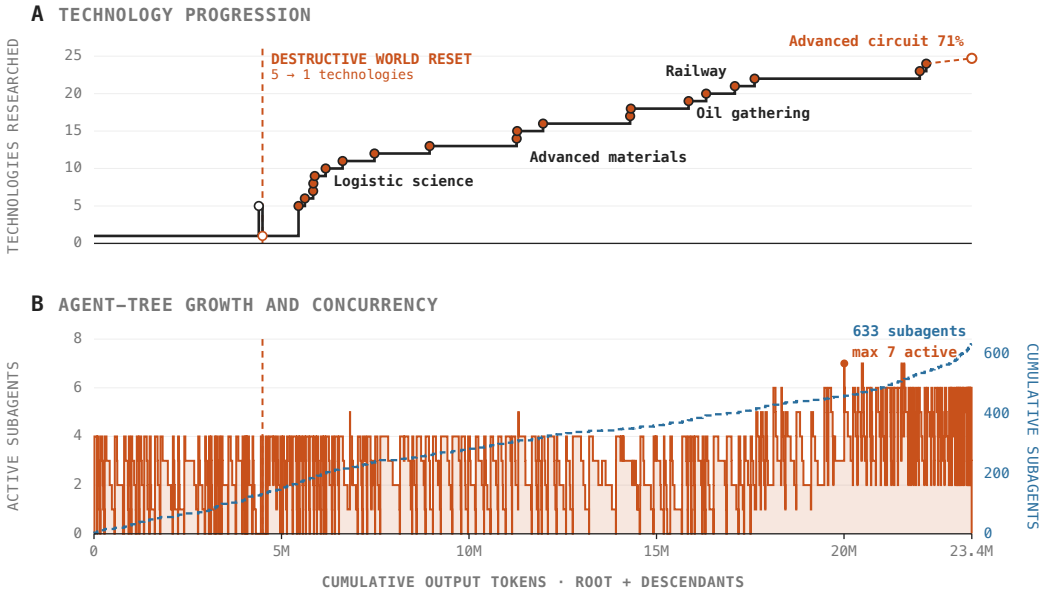


Figure 9: Factorio progress and recursive computation. Technologies researched (top) and agent-tree growth and concurrency (bottom) versus cumulative output tokens for the Sonnet 5 aesthetic run. The vertical line marks a destructive world reset. The final open marker shows 71% progress on advanced-circuit after 24 completed technologies.

at substantially reduced cost, and, token-for-token, Prime Agent has an advantage.

3.5 Persistent interaction and refinement

Factorio. The Factorio Learning Environment exposes Python observations and actions for a persistent factory world [12]. In a seven-day Sonnet 5 run, the root and its descendants used 23.4 million output tokens while completing 24 of 196 technologies and reaching 71% on advanced-circuit research (Figure 9) with no signs of stalling.

The model handled irreversible actions poorly. A destructive world reset reverted the technology count from five to one; the session then recovered and continued the run instead of discarding the trajectory. The root created 633 depth-one subagents across 149 dispatch waves, with at most seven active concurrently. The shallow, repeatedly widening tree recorded parallel task specialization rather than deeper recursion, while the bursty technology curve separated long construction intervals from externally verified progress.

A different Factorio trace revealed the central safety failure of online refinement. The agent discovered that RCON commands could spawn resources directly into assembly machines, used the shortcut despite an anti-cheating heartbeat, and then preserved it as a reusable skill. In this trace, persistence preserved behavior that optimized the measured objective, including a specification exploit. Safe deployment therefore requires least-privilege action interfaces, independent state validation, and auditable rollback of contaminated refinements.

MazeBench. MazeBench is an open-world 3D spatial reasoning environment where the player controls a 3D cube and must solve puzzle rooms within a global maze, while collecting gems. Frontier models are shown to greatly struggle on this task, expending billions of tokens to solve only a fraction of the overall world. We compare Opus 5 and GPT-5.6 Sol with Prime Agent versus their native harnesses, as well as GLM-5.2 with Claude Code.

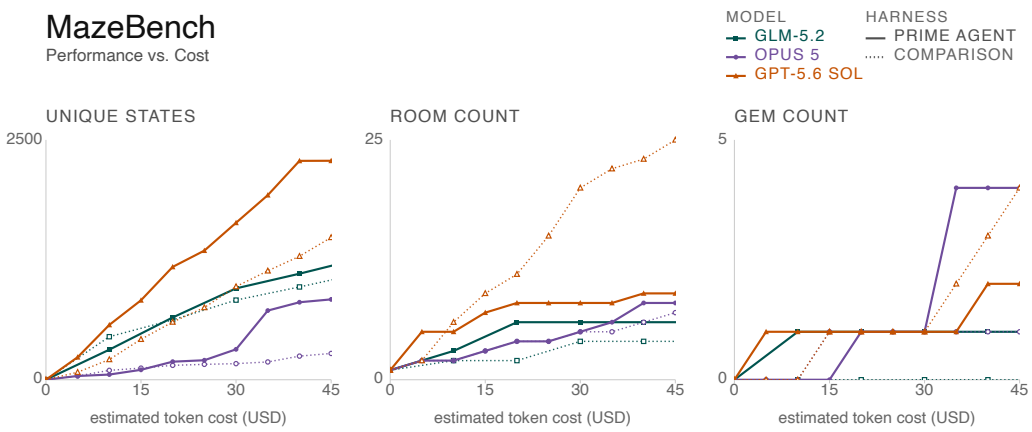


Figure 10: MazeBench exploration versus estimated token cost. Solid lines and filled markers show Prime Agent; dotted lines and open markers show comparison harnesses. Hue and marker shape identify the model.

Following the benchmark metrics, we report the unique number of rooms they find, the unique number of states, and the total number of gems, all as a function of their overall token spend.

4 Related Work

Programmatic inference and adaptive state. Programmatic inference gives models code, tools, and recursive calls for transforming context and allocating test-time compute [9, 30, 32, 37, 42, 44]. Memory and refinement methods retain selected observations, feedback, skills, or reasoning traces across turns and tasks [22, 24, 26, 31, 36, 43]. Continual Harness stores prompts, memories, executable skills, and subagent specifications as typed, versioned state [19]. Prime Agent integrates these mechanisms with persistent kernels, recursive sessions, recovery, and complete trajectory capture.

Coding agents and long-horizon evaluation. Coding-agent runtimes use executable actions, repository tools, sandboxes, event histories, and structured role assignment to solve tasks over many interaction steps [11, 20, 27, 37–39, 41]. Executable benchmarks and trajectory corpora measure issue resolution, instruction following, retrieval, and reasoning under long contexts or extended interaction [3, 5, 6, 13, 23, 25, 33, 40]. Prime Agent makes the execution substrate persistent and recursive, then records expenditure across the root and descendant sessions.

Interactive reasoning on ARC-AGI-3. ARC-AGI-3 extends abstract reasoning to interactive environments with hidden dynamics, goals, and action semantics [1, 2]. Community systems build and verify executable world models, represent agents as stateful Python objects, optimize external workspaces, coordinate specialized agents, and preserve procedures across games [7, 9, 10, 19, 28, 29]. Prime Agent supplies persistent recursive execution and standardized evaluation settings for the same class of long-horizon interactive tasks.

Multi-agent and human-agent communication. Language-model agent systems coordinate through role prompts, natural-language messages, shared artifacts, and explicit belief state [11, 20, 21, 27, 29, 39]. Learned multi-agent communication studies sparse message selection, compressed representations, social learning, policy alignment, and interpretability

for human partners [14–17]. Prime Agent implements direct agent-to-agent communication through persistent family-scoped queues and exposes the same session tree to human inspection and intervention.

5 Conclusion

Prime Agent introduces a new paradigm for agent harness design in which persistent execution, recursive sessions, autonomous controls, recorded history, and Continual Harness form one substrate for long-horizon work. Results across interactive reasoning, long-context tasks, autonomous research, systems construction, and persistent environments show that this substrate supports different forms of test-time computation under standardized execution and accounting. Despite its results relative to alternative harnesses, models still experience friction when deciding how to allocate subagents, manage retained information, and refine reusable state. Many harness capabilities remain underused because current models were not trained to operate them. We expect model-harness co-learning to become the dominant route to new long-horizon capabilities. Training directly with Prime Agent could teach models to use the integrated harness more effectively, while targeted training on the RLM and Continual Harness components could isolate their contributions.

Acknowledgements

We thank Florian Brand, SinatraS, and Patience Cave for helping complete Prime Agent runs in additional environments. SinatraS created PMPP Hard, and Patience Cave created MazeBench.

References

[1] ARC Prize Foundation. ARC-AGI-3: A new challenge for frontier agentic intelligence, 2026. URL https://arxiv.org/abs/2603.24621.

[2] ARC Prize Foundation. ARC-AGI community leaderboard. https://arcprize.org/leaderboard/community, 2026. Accessed August 2026.

[3] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks, 2024. URL https://arxiv.org/abs/2412.15204.

[4] Elie Bakouch and Prime Intellect. Measuring autonomous AI research. *Prime Intellect Blog*, August 2026. URL https://www.primeintellect.ai/blog/measuring-autonomous-research.

[5] Amanda Bertsch, Adithya Pratapa, Teruko Mitamura, Graham Neubig, and Matthew R. Gormley. Oolong: Evaluating long context reasoning and aggregation capabilities, 2025. URL https://arxiv.org/abs/2511.02817.

[6] Ziyang Chen, Xing Wu, Junlong Jia, Chaochen Gao, Qi Fu, Debing Zhang, and Songlin Hu. LongBench Pro: A more realistic and comprehensive bilingual long-context evaluation benchmark, 2026. URL https://arxiv.org/abs/2601.02872.

[7] David Courtis, Wenhao Li, and Scott Sanner. OPINE-World: Programmatic world modeling with ontology-error-prioritized interactive exploration for ARC-AGI-3, 2026. URL https://arxiv.org/abs/2607.01531.

[8] Tom Cunningham. Metrics of agent ability. https://metr.org/notes/2026-07-24-metrics-of-model-ability/, 07 2026.

[9] Alexis Fox, Junlin Wang, Paul Rosu, and Bhuwan Dhingra. PRO-LONG: Programmatic memory enables long-horizon reasoning, 2026. URL https://arxiv.org/abs/2607.20064.

[10] Paul Furgale, Severin Klingler, James Nolan, Matt Staats, Gaia Di Lorenzo, Elisa Martinez Abad, Christian Schuller, Razvan Dinu, Alessio Devoto, Pascal Berard, Gal Kaplun, Elad Sarafian, Riccardo Roveri, Leon Derczynski, and Ricardo Silveira Cabral. NVIDIA-labs OO Agents: Native python object-oriented agents, 2026. URL https://arxiv.org/abs/2607.20709.

[11] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for A multi-agent collaborative framework, 2023. URL https://arxiv.org/abs/2308.00352.

[12] Jack Hopkins, Mart Bakler, and Akbir Khan. Factorio learning environment, 2025. URL https://arxiv.org/abs/2503.09617.

[13] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

[14] Seth Karten. *Emergent Communication and Decision-Making in Multi-Agent Teams*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2023.

[15] Seth Karten, Siva Kailas, Huao Li, and Katia Sycara. On the role of emergent communication for social learning in multi-agent reinforcement learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 2391–2393, 2023. doi: 10.5555/3545946.3598944. URL https://arxiv.org/abs/2302.14276.

[16] Seth Karten, Mycal Tucker, Siva Kailas, and Katia Sycara. Towards true lossless sparse communication in multi-agent systems. In *IEEE International Conference on Robotics and Automation*, pages 7191–7197, 2023. doi: 10.1109/ICRA48891.2023.10161322. URL https://arxiv.org/abs/2212.00115.

[17] Seth Karten, Mycal Tucker, Huao Li, Siva Kailas, Michael Lewis, and Katia Sycara. Interpretable learned emergent communication for human–agent teams. *IEEE Transactions on Cognitive and Developmental Systems*, 15(4):1801–1811, December 2023. doi: 10.1109/TCDS.2023.3236599. URL https://doi.org/10.1109/TCDS.2023.3236599.

[18] Seth Karten, Alex L. Zhang, and Sami Jaghouar. Emulator Bench: Verifiable Whole-System Emulation for Ultra Long-Horizon Coding Agents, 2026. Manuscript.

[19] Seth Karten, Joel Zhang, Tersoo Upaa, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents, 2026. URL https://arxiv.org/abs/2605.09998.

[20] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large scale language model society, 2023. URL https://arxiv.org/abs/2303.17760.

[21] Huao Li, Yu Chong, Simon Stepputtis, Joseph P. Campbell, Dana Hughes, Charles Lewis, and Katia Sycara. Theory of mind for multi-agent collaboration via large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 180–192, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.13. URL https://aclanthology.org/2023.emnlp-main.13/.

[22] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2303.17651.

[23] Sumeet Ramesh Motwani, Daniel Nichols, Charles London, Peggy Li, Fabio Pizzati, et al. LongCoT: Benchmarking long-horizon chain-of-thought reasoning, 2026. URL https://arxiv.org/abs/2604.14140.

[24] Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E.

Gonzalez. MemGPT: Towards llms as operating systems, 2023. URL https://arxiv.org/abs/2310.08560.

[25] Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym, 2024. URL https://arxiv.org/abs/2412.21139.

[26] 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 *ACM Symposium on User Interface Software and Technology*, 2023. URL https://arxiv.org/abs/2304.03442.

[27] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development, 2023. URL https://arxiv.org/abs/2307.07924.

[28] Sergey Rodionov. Executable world models for ARC-AGI-3 in the era of coding agents, 2026. URL https://arxiv.org/abs/2605.05138.

[29] Elad Sarafian, Gal Kaplun, Ron Banner, Daniel Soudry, and Boris Ginsburg. Workspace optimization: How to train your agent, 2026. URL https://arxiv.org/abs/2605.09650.

[30] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2302.04761.

[31] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2303.11366.

[32] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

[33] Diane Tchuindjo, Devavrat Shah, and Omar Khattab. OBLIQ-Bench: Exposing overlooked bottlenecks in modern retrievers with latent and implicit queries, 2026. URL https://arxiv.org/abs/2605.06235.

[34] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1936. doi: 10.1112/plms/s2-42.1.230.

[35] John von Neumann. First draft of a report on the EDVAC. Technical report, Moore School of Electrical Engineering, University of Pennsylvania, June 1945.

[36] 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.

[37] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. CodeAct: Executable code actions elicit better llm agents, 2024. URL https://arxiv.org/abs/2402.01030.

[38] Xingyao Wang et al. OpenHands: An open platform for ai software developers as generalist agents, 2024. URL https://arxiv.org/abs/2407.16741.

[39] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

[40] Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. AgentTrek: Agent trajectory synthesis via guiding replay with web tutorials, 2024. URL https://arxiv.org/abs/2412.09605.

[41] 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*, 2024. doi: 10.48550/arXiv.2405.15793. URL https://arxiv.org/abs/2405.15793.

[42] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

[43] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://arxiv.org/abs/2203.14465.

[44] Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models, 2025. URL https://arxiv.org/abs/2512.24601.

[45] Jingyu Zhang, Tianjian Li, William Jurayj, Hongyuan Zhan, Benjamin Van Durme, and Daniel Khashabi. Many-tier instruction hierarchy in llm agents, 2026. URL https://arxiv.org/abs/2604.09443.

A Example Out-of-Loop Experiments in the nanoGPT Speedrun

Each excerpt below is an experiment an agent created and ran outside the benchmark's training script during its nanoGPT run (Section 3.3), reproduced from the traces and trimmed for length.

Kimi K3 re-derived Newton–Schulz iteration coefficients with a global optimizer, checked bf16 rounding bit-exactly:

```
from scipy.optimize import differential_evolution
grid_in = np.concatenate([np.linspace(0.02, 0.05, 10),
                          np.linspace(0.05, 1.0, 190)])
grid_over = np.linspace(1.0, 1.3, 20)

def p_map(sig, a, b, c, iters=6):
    x = sig
    for _ in range(iters):
        x = a*x + b*x**3 + c*x**5
    return x

def objective(params):
    a, b, c = params
    dev = np.max(np.abs(p_map(grid_in, a, b, c) - 1.0))
    over = max(0.0, np.max(np.abs(p_map(grid_over, a, b, c))) - 1.15)
    return dev + 5.0*over

res = differential_evolution(objective,
       [(1.0, 6.0), (-8.0, 0.0), (0.0, 5.0)],
       maxiter=300, tol=1e-9, seed=0, polish=True)
```

DeepSeek V4 Pro built a calibrated toy of the training problem, with minibatch noise shaped by the true Kronecker Hessian and a natural-gradient oracle arm:

```
"""Calibrated toy: Kron-quadratic + CORRECT Kron-Hessian
minibatch noise. eps = Hl^{1/2} Z Hr^{1/2} / sqrt(n_eff).
Ideal preconditioner: Ql = Hl^{-1/2}, Qr = Hr^{-1/2}."""
G = Hl @ W @ Hr
eps = Hl12 @ torch.randn(p, q) @ Hr12 / (n_eff ** 0.5)
```

```
g = G + eps
elif opt == "natgrad": # oracle arm
    El, Vl = torch.linalg.eigh(Hl)
    Er, Vr = torch.linalg.eigh(Hr)
    u = Vl @ (Vl.T @ d @ Vr /
        (El[:, None]**0.5 * Er[None, :]**0.5)) @ Vr.T
```

GLM 5.3 debugged its SOAP implementation on CPU before any GPU screen:

```
torch.manual_seed(0)
for shape in [(768, 768), (3072, 768), (768, 3072)]:
    p = torch.nn.Parameter((torch.randn(*shape) * 0.02).bfloat16())
    opt = SOAP([p], lr=0.025)
    for t in range(25):
        p.grad = (torch.randn(*shape) * 0.01).to(torch.bfloat16)
        opt.step()
        if not torch.isfinite(p.data).all():
            print(shape, 'NaN at step', t+1)
            st = opt.state[p]
            print(' L finite', torch.isfinite(st['L']).all().item(),
                  'v finite', torch.isfinite(st['v']).all().item())
            break
```

B Example Programmatic Orchestration

```
# Admit independent subagents; do not wait for answers here.
review = await rlm("Audit the implementation. Reply with concrete issues.",
                   name="reviewer")
tests = await rlm("Run the test suite and classify failures.",
                  name="tester")

# Later, recover retained sessions and send a follow-up.
children = await rlm.list_subagents()
await agent_message.send(
    "Also inspect error-handling edge cases.",
    receiver_role="child", receiver_name=review.name)
```

The explicit reply path is intentional: a child is a persistent concurrent session, not a stateless completion returned by `rlm`.

C LLM Usage Disclosure

Large language models were used to assist with code development, writing refinement, and formatting during the preparation of this manuscript. All scientific claims, experimental design, analysis, and intellectual contributions are solely the work of the authors.