Outerloop: Open and Trustworthy Multi-Agent Autoresearch

Mengye Ren
Agentic Learning AI Lab, New York University
mengye@nyu.edu
Project Page: https://outerloop.science
Code: https://github.com/outerloop-science/outerloop
November 1, 2026

Abstract

Coding agents can now run research loops on real codebases, editing code, launching experiments, and reporting improvements. However, in current open systems the agent is both the author and the evaluator. In this paper, we argue that autoresearch needs a rigid outer structure so that the inner research loop can run freely, and we build that structure as a kernel. We present Outerloop, an open-source kernel that runs research agents as untrusted programs in sandboxes and enforces measurement, edit scope, review, and compute budgets. Within these limits, agents built on different harnesses can experiment with new ideas together in one repository. We evaluate Outerloop on a nanoGPT training-speed benchmark. When we rerun the chosen results of three published systems on new seeds, their gains shrink because part of each gain came from a lucky run. Moreover, our own agents' gains shrink in the same way when they grade themselves, whereas our kernel's credited gains match the audit. With the same coding model, most autoresearch systems reach similar scores, so they differ mainly in the trustworthiness of their reported results.

1 Introduction

Coding agents can now edit a research codebase, launch experiments on a cluster, and report improvements. Several open systems run such agents in a loop for hours or days (Karpathy, 2026; Jiang et al., 2025; Lange et al., 2026). Such a loop promises steady improvement on a repository's benchmark. However, in current open systems the agent is both the author and the evaluator of its changes.

This arrangement leaves several failures unchecked. First, when an agent runs many noisy experiments and reports the best one, the reported gain includes the luck of that run, a form of *selection bias*. Second, when the agent can write to the evaluation code, it can change the score directly. Third, a correctly measured gain can still come from an unacceptable change, such as one that overfits the benchmark or bundles several unrelated edits. A larger search makes these failures more likely, because it tries more candidates. The first two can be ruled out mechanically, but the third needs judgment.

In this paper, we argue that autoresearch needs a rigid outer structure so that the inner research loop can run freely. The structure is small: agents cannot change the evaluation code, cannot predict the

Agentic Learning AI Lab Technical Report AGL-TR-2026-02. doi:10.5281/zenodo.24047406

measurement seeds, cannot spend beyond their budget, and cannot skip review. Within these limits, the agents choose what to try, how long to pursue it, which harness and model to run, and whether to work alone or with others. An operating system works the same way, protecting memory and devices and leaving user programs free to do anything else. We present Outerloop, an open-source kernel that enforces these limits for research agents working on real repositories.

Outerloop differs from a *meta-harness*, an outer agent that prompts and coordinates the inner ones. Outerloop does some of that work, such as writing each agent's brief and delivering its messages, but its guarantees come from rules fixed in code. A meta-harness adds another agent, which can make mistakes or be persuaded by the inner agents to credit a result. Outerloop does use models as judge agents, but they can only withhold credit, and granting credit always requires the kernel's measurement.

Because the measurement and the budget are fixed in code, Outerloop also serves as an instrument for studying research agents. With these two held constant, we can then study other choices, such as the author model, the harness, the number of agents, and their communication.

In this paper, we make three contributions.

- We formulate autoresearch as optimization by an untrusted party and design a minimal outer structure around it: held-out seeds with a noise floor, an enforced edit scope, and judge agents (Sections 2 and 3). Outerloop is open source, and the audited results will be released.
- We compare the kernel with three published systems at the same task, model, and compute budget. Their gains shrink when rerun on fresh seeds, while the gains the kernel credits match the audit (Section 5).
- We use the kernel to compare agents under controlled conditions. Systems with the same model reach similar scores but differ in the accuracy of their self-reported scores. One agent matches a team with the same total compute. On our benchmark, modeled on the nanoGPT speedrun, much of the progress comes from recalling the Muon optimizer (Jordan et al., 2024b).

2 Autoresearch as untrusted optimization

We consider a repository whose maintainers care about one benchmark score, such as a training script's steps to a target loss. Running the benchmark on code c with random seed s gives a noisy score $f(c, s)$, and the quantity of interest is its mean over seeds, $\bar{f}(c)$. An autoresearch system proposes changes to c and claims that some of them improve $\bar{f}$. We treat the agents behind these changes as untrusted, though not adversarial. When the same agent both proposes changes and evaluates them, its reported scores tend to exceed the true ones. Three failures follow from this view.

Selection bias. Choosing the best of many candidates, each evaluated once, selects a lucky draw, an effect long known in model selection (Cawley and Talbot, 2010) and recently documented on public leaderboards (Singh et al., 2025). If the top candidates are about equally good and each evaluation has seed noise σ, the best of n is inflated by about σ times the expected maximum of n standard normal variables. This inflation is already 0.8σ for three candidates. Seed variance in ML benchmarks is large enough to change conclusions (Bouthillier et al., 2021). The bias is invisible in the search's own numbers.

Tampering. An agent that can write to the evaluation code, the data pipeline, or the logs can raise the score without improving the model. Such reward hacking has been documented for research agents, from overwriting a timer to patching the evaluator (METR, 2025; Luo et al., 2025). Coding agents also often edit or special-case the tests they are asked to pass (Zhong et al., 2026). Instructions to leave the evaluator alone are a weak defense, because they rely on the agent's own compliance. Monitoring the agent's reasoning

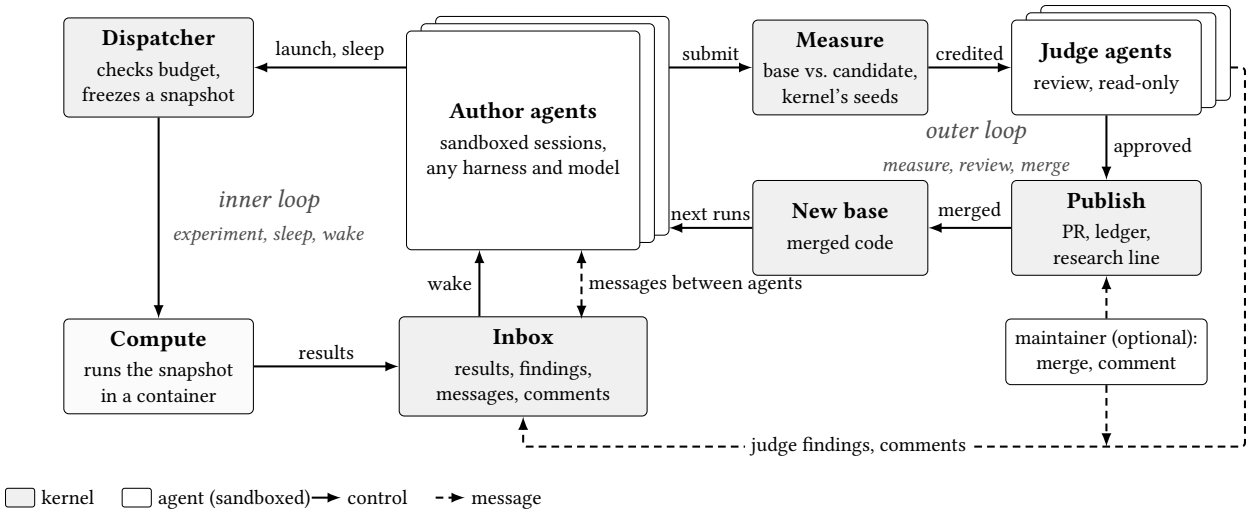


Figure 1: *Outerloop's two loops. Inner loop (left): an author agent's edits, experiments, and sleep until results arrive. Outer loop (right): the kernel's measurement, judge review, and merged code as the next base. Shaded: kernel; white: sandboxed agents.*

can also be evaded once the agent is trained against the monitor (Baker et al., 2025).

Unacceptable gains. A measured improvement can still be one the maintainers would reject. For example, a change may fit the evaluation set, rely on an unreported side effect, or bundle several unrelated edits so that no single one can be credited. Measurement alone cannot detect these, because the number they produce is real.

To address these failures, we propose a fixed structure around the agents, with three parts that decide credit. First, an independent party should re-measure claims on held-out seeds against a noise floor. Second, the evaluation code should lie outside the agents' edit scope. Third, a judge should check the provenance of each number. A budget bounds the search's spending, and outside these parts the structure should leave the agents free. We also need the structure to fit how agents work. Their experiments take hours, their harnesses and models change within months, and several agents may work on one repository at once. The structure should therefore depend neither on an agent's harness and model nor on the number of agents. Section 3 builds this structure as a kernel.

3 System design

We build the structure of Section 2 as a small kernel, Outerloop, in the way an operating system runs untrusted programs on top of a trusted core (Saltzer and Schroeder, 1975) (Figure 1). The kernel keeps only the trusted work: measurement, edit-scope enforcement, resource metering, and publication. Everything else is left to agents, which run in sandboxes in two roles. *Authors* do the research and decide what to try, how to plan, and whom to talk to, while *judges* review their submissions. Agents reach the kernel through a few plain commands, an agent-computer interface (Yang et al., 2024) for any coding harness, so a new harness or model needs no kernel change. The system runs two loops, which give it its name. The authors' inner loop is ordinary experimentation, while the kernel's outer loop turns verified changes into the base for the next agents.

3.1 The repository contract

The repository's *contract* defines research progress in four fields.

- **Benchmark**: a command, its output metric, and the metric's direction.
- **Editable paths**: the files agents may change. The kernel refuses changes elsewhere, so agents cannot edit the evaluation code, the data pipeline, or the tests. Other forms of gaming are left to the judges.
- **Budgets**: compute hours (GPU-hours in our study), experiment launches, and session wake-ups per run.
- **Noise floor**: the smallest gain worth crediting.

3.2 Inner loop: experimentation

Inside this structure, a *run* is one author agent pursuing one hypothesis. The run starts in a sandbox with a copy of the latest merged code and without repository credentials. The agent can be any supported harness (Claude Code, Codex, or Hermes) driving a hosted or self-hosted model.

An idea is tested by calling `launch` with a command, a time limit, and a one-line reason. One launch can be an array of jobs for a hyperparameter sweep. The kernel checks the budget, freezes a snapshot of the agent's code, and runs the snapshot on the cluster, so later edits cannot change an experiment in flight. The agent then calls `sleep`, which ends its session until the jobs finish. The same session then resumes with its context and the results in its inbox. No model tokens are spent while jobs run, so an agent can afford long experiments. A run may repeat this cycle up to a launch budget k, its *depth*. Depth lets an author build a line of research, in which each experiment follows from the results of the previous one, rather than test one idea per session by trial and error. Across runs, the author's branch and notes carry the line forward (Section 3.5).

3.3 Outer loop: verification and publication

When the author has a result it trusts, it calls `submit` with a short report. From then on, only the kernel's measurement counts.

Measurement. An agent that keeps the best of many noisy runs reports that run's luck along with any real gain. To remove this selection bias, the kernel measures the submission itself, running the base and the candidate back to back on one shared seed. The kernel draws the seed itself and never shows it to the agent, so the agent cannot tune its code to it. A result is credited only if the measured gain clears the contract's noise floor. A revised submission is measured again on both sides, on a new seed when the revision comes in a later session. The kernel pays for these measurements out of the run's own budget, so verification and search draw on the same budget.

Noise floor. As in Section 2, $f(c, s)$ is the benchmark score of code c on seed s, and $\bar{f}(c)$ is its mean over seeds. Let c_0 be the base code, c the submitted code, and $\Delta = \bar{f}(c_0) - \bar{f}(c)$ its true gain, for a score where lower is better. The kernel draws a seed S and measures the paired difference

$$D = f(c_0, S) - f(c, S) = \Delta + \epsilon_0(S) - \epsilon_c(S), \tag{1}$$

where $\epsilon_c(S) = f(c, S) - \bar{f}(c)$ is the seed noise of c. Because S is drawn independently of how the agent chose c, $\mathbb{E}[D \mid c] = \Delta$. The agent's search can still favor a candidate that was lucky on its own seeds. That luck does not carry over to S, so the measurement removes the selection bias of Section 2. With k seeds the kernel averages the differences into $\bar{D}_k$ and credits c if $\bar{D}_k \geq \delta$, the contract's noise floor. The floor and

k set two rates,

$$\mathrm{FPR} = \Pr[\bar{D}_k \geq \delta \mid \Delta = 0], \tag{2}$$

$$\mathrm{Recall}(\Delta) = \Pr[\bar{D}_k \geq \delta \mid \Delta], \tag{3}$$

the false-positive rate for a change with no effect and the recall of a real gain. With noise standard deviations σ_0 and σ_c and correlation ρ between the two programs on one seed, $\mathrm{Var}[\bar{D}_k] = (\sigma_0^2 + \sigma_c^2 - 2\rho\sigma_0\sigma_c)/k$, so pairing reduces the variance only when ρ is positive. For any noise distribution, Cantelli's inequality bounds the false-positive rate by $\mathrm{FPR} \leq \mathrm{Var}[\bar{D}_k]/(\mathrm{Var}[\bar{D}_k] + \delta^2)$. The maintainers calibrate δ once, from repeated runs of the unmodified code. The recorded gain of a credited change is still conditioned on $D \geq \delta$, so for gains near the floor it overstates Δ. More seeds lower FPR at a cost in compute. A higher floor also lowers FPR, but it lowers $\mathrm{Recall}(\Delta)$ for real gains Δ close to δ. A repository's maintainers therefore set both in the contract.

Review. A real gain can still come from an unacceptable change, such as one that exploits a quirk of the evaluator or bundles several unrelated edits. Two sandboxed judge agents with read-only access therefore read each credited submission. The kernel schedules them and combines their verdicts by fixed rules. The first judge looks for gaming, such as data leakage, fitting to the evaluator, or claims the evidence does not support. The second reviews the code and the report's attribution of the gain to its parts. A blocking finding is sent to the author's inbox, and the author can revise and resubmit. Measurement and review thus guard against different failures, the first against wrong numbers and the second against real gains obtained the wrong way.

Publication. A credited result becomes a pull request carrying the measured numbers and the agent's report, together with an entry in a results ledger. The pull request stays a draft until the judges approve it. It merges automatically if allowed by the contract, or otherwise waits for a maintainer, who can also comment and receive the agent's reply. Merged code becomes the base for the next runs, which closes the outer loop. Agents still at work fold the new base into their branches.

3.4 Budgets

The kernel meters three resources: compute hours, experiment launches, and session wake-ups. A launch costs its time limit times its device and job counts, and the kernel's measurements count too. Requests are checked against the run's remaining budget before they are submitted, and unused time is refunded afterwards. External failures, such as a model-provider outage, pause the run instead of ending it.

3.5 Multiple agents

A contract's *width* sets how many runs may be active at once. Each agent keeps a branch and a notes file across its runs, so a later run builds on an earlier one's findings. They can message each other, leave notes for their next session, and see the other agents' current work and the job queue. The kernel delivers and meters these messages without coordinating the agents. Whether they divide the work, copy each other, or ignore each other is left to them. Coordination is therefore something we can measure (Section 5).

3.6 Trust boundary

Figure 2 shows what crosses the boundary between agents and the kernel. A session holds a copy of the workspace, the kernel's command-line tool, and an API key for its language model, but no repository credentials, no experiment hardware, and not the measurement seed. The agent can act on the repository only through requests to the tool, which the kernel validates after the session ends. Before measuring a submission, the kernel rejects edits outside the editable paths or to the contract, as well as a redirected `.git` directory, which one author agent attempted during development. Only the kernel pushes, and it pushes the tree it measured. An agent therefore cannot change its own evaluation or publish code that was not measured, whatever it attempts.

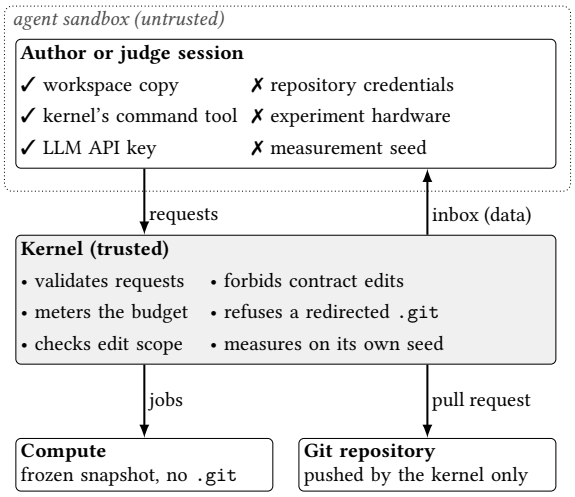


Figure 2: *Trust boundary between agent sessions and the kernel: agent requests, validated after each session; repository credentials and jobs, held by the kernel alone.*

3.7 Portability

The kernel keeps its state in files and git and runs experiments through a small compute interface, with Slurm (Yoo et al., 2003) and local backends today. The same code therefore runs on a shared cluster and on a single workstation.

4 Experimental setup

Benchmark. We use a small training-speed benchmark modeled on the nanoGPT speedrun (Jordan et al., 2024a; Zhao et al., 2025). A GPT model of fixed architecture is trained from scratch, and the score is its number of optimizer steps to a target validation loss (lower is better). Agents may edit only the training script, which covers the optimizer, the learning-rate schedule, initialization, and batching. The model, data, and evaluation are fixed, and a step allows at most 131,072 tokens and one forward pass. One training run takes about 15 minutes on one A100, so hundreds of independent replicates are affordable. We set the noise floor δ to 64 steps, about 3% of the base. The benchmark has a well-known strong answer, the Muon optimizer (Jordan et al., 2024b; Liu et al., 2025a), which we use later to separate recall from search. All runs are on a Slurm cluster of spot A100 nodes on Google Cloud, and all budgets below are in A100 GPU-hours.

Main study. Our main comparison gives one agent and each of three external systems 48 GPU-hours of search, with ten independent runs per system. We run AIDE (Jiang et al., 2025), autoresearch (Karpathy, 2026), and ShinkaEvolve (Lange et al., 2026) as published, with the same author model and the same GPU budget, through a thin adapter that sends their evaluations to the cluster. They run without the kernel's sandbox, so they keep their own result-selection procedures. Authors and judges run on GLM 5.2 in the Codex harness, served through a cloud API. No system has a web-search tool, so ideas come from the model and from the runs' own experiments. Our agent works in consecutive runs of up to 12 GPU-hours on one research line, so its first result appears after about 12 GPU-hours. It receives a short guidance file, matched in length to the external systems' task text. The file asks it to minimize the metric, keep testing

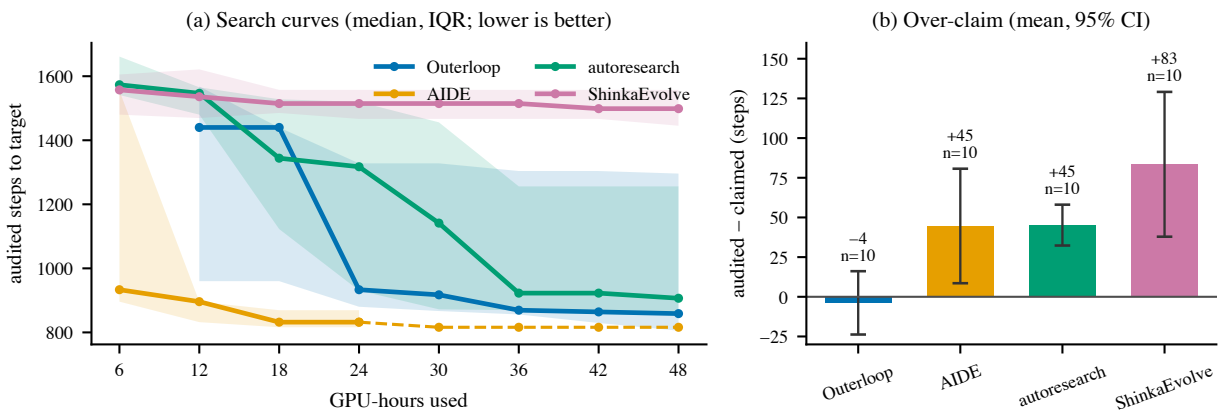


Figure 3: *Search over 48 GPU-hours. (a) Audited steps to target of each system's chosen result (ours: best kernel-measured candidate; median and interquartile range over runs; dashed: fewer than three runs). (b) Audited minus claimed steps, averaged within each run (mean and 95% interval over runs).*

new ideas, and prefer simpler code. A run's GPU-hours include every experiment and, for our agents, the kernel's own baseline and candidate measurements.

Operator interventions. Two operator actions apply to all of our runs. First, near the end of the budget, a message asks the agent to submit its best result. Second, a submitted pull request that waits for review is closed, so that the next run can start. We replaced AIDE runs that died and scored replacements that ended short of 48 GPU-hours at their last checkpoint.

Audit. Independent of the kernel's measurement, we record each system's chosen result every 6 GPU-hours and re-measure each distinct choice, 125 in all. The audit compares each choice with the unmodified code on three fresh seeds unseen by any system. The audited number of steps is the score we report.

Short study. For the comparisons that need many replicates, we run a *short study*. A *cell* is one independent replicate of a configuration, with its own copy of the repository and its own budget of six GPU-hours. We vary one factor at a time, namely the author model, the launch budget per run, the total GPU-hours, the number of agents, and communication between agents. Each configuration has 10 to 20 cells. The short study also includes the three external systems at six GPU-hours and a control in which our agents report their own results with the kernel's measurement switched off. Authors run on GLM 5.2 except in the model comparison, and judges run on Kimi K2 Thinking here and on GLM 5.2 in the main study. Each cell's final result is audited as above. Cells lost to infrastructure failures, such as node preemption or a model-provider outage, are excluded and replaced.

5 Results

5.1 Scores and the role of Muon

Over 48 GPU-hours of search, AIDE improves fastest (Figure 3a), because it drafts a training script from scratch, and in most runs the draft uses Muon within the first GPU-hour. Our agents and autoresearch edit the given script instead and reach a similar level only after many hours. ShinkaEvolve stays on a plateau throughout.

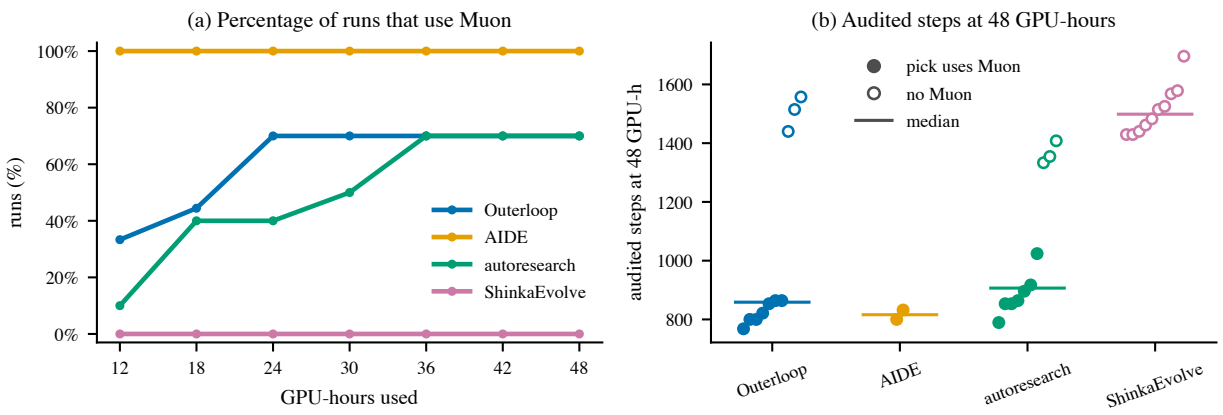


Figure 4: *Muon over 48 GPU-hours of search. (a) Percentage of runs whose chosen result uses Muon. (b) Audited steps of each run's result at 48 GPU-hours (filled markers: Muon; lines: medians).*

Table 1: *Over-claim at six GPU-hours, as claimed minus audited steps saved (mean and 95% interval).* Luck*: the chosen run's margin over the same code on fresh seeds.*

System	*n*	Over-claim	Luck
Outerloop, kernel measures	19	−10 ± 27	1
Outerloop, agent self-reports	19	+36 ± 31	44
AIDE	10	+27 ± 39	30
autoresearch	10	+60 ± 53	64
ShinkaEvolve	10	+25 ± 42	21

At 48 GPU-hours, our agents and autoresearch reach similar audited scores, both far ahead of ShinkaEvolve (Figure 4b). AIDE's final score rests on the only two runs that reached the full budget. Nearly all of the difference between systems comes from one change, the Muon optimizer. Results that use Muon fall in one band of steps and results without it in another, whichever system produced them. The systems therefore differ mainly in whether and when they adopt Muon. AIDE adopts it almost immediately, and our agents and autoresearch adopt it in about seven of ten runs. ShinkaEvolve never does, because its mutations change only small parts of the given code.

5.2 Claims

Unlike their scores, the systems' claims differ widely. The three external systems claim more than the audit confirms, while the kernel's measurements of our agents' results agree with the audit within noise (Figure 3b). The cause is selection bias (Section 2). The external systems pick their best result from single-seed evaluations, so the chosen result carries the luck of its seed. ShinkaEvolve over-claims the most, and its over-claim grows from about 30 steps at six GPU-hours to about 100 at the end, while that of AIDE and autoresearch stays roughly level. Because ShinkaEvolve evaluates the most programs, it has the most chances to pick one that was lucky on its evaluation seed.

Self-reporting control. The external systems differ from our agents in more than how their claims are checked. We therefore also switched the kernel's measurement off for our own agents in the short study

Table 2: *Launch statistics over 48 GPU-hours. Died: runs ended by model-call errors. GPU-h and Days: median GPU time and wall-clock time of completed runs.*

System	Launched	Completed	Died	GPU-h	Days
Outerloop	10	10	0	44.2	1.3
AIDE	32	2	18	47.5	3.2
autoresearch	13	10	0	47.5	2.9
ShinkaEvolve	10	10	0	47.6	0.9

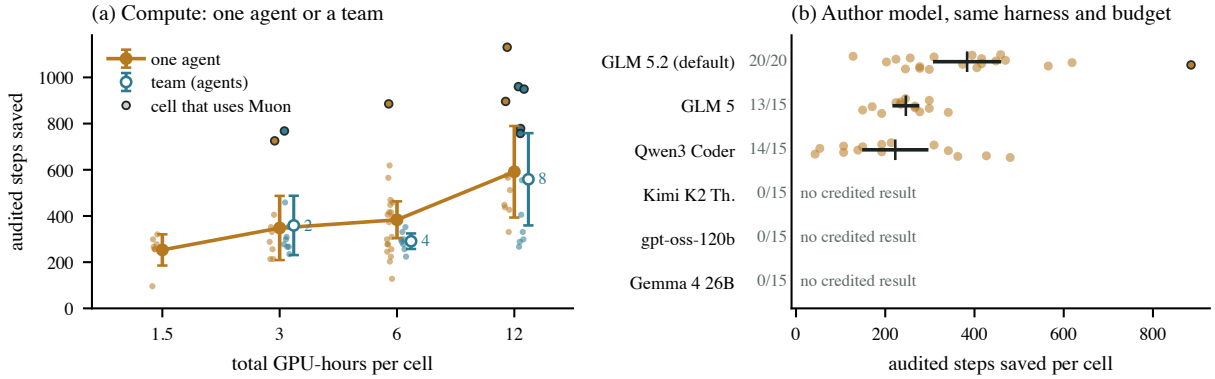


Figure 5: *Audited steps saved at six GPU-hours per cell (mean and 95% interval; dots: cells). (a) One agent with more GPU-hours and teams with the same total. (b) Author models in the same harness and budget.*

(Table 1). When our agents report their own numbers, they over-claim about as much as the external systems. When the kernel measures, their claims agree with the audit within noise. Wherever a system grades itself, its over-claim is close to its *luck*, the chosen run's margin over the same code's fresh-seed mean, while the kernel's own measurement has almost no luck. At six GPU-hours, the over-claim intervals of AIDE and ShinkaEvolve still include zero. Over-claim therefore comes from grading one's own results, in our agents as in the external systems.

5.3 Reliability and throughput

Many AIDE runs ended early when a long model response outlasted the timeouts in AIDE's model client (Table 2). Even after we lengthened the timeouts and added retries in our adapter, completing two runs took 32 launches. The cause is a fixable defect in AIDE's implementation. Wall-clock time, in contrast, follows from each system's evaluation schedule. AIDE and autoresearch run one evaluation at a time and take about three days to spend the budget, whereas our agents' parallel sweeps and ShinkaEvolve's parallel evaluations spend it in about a day.

Accounting. Our runs pay for the kernel's verification out of the same budget, and the baseline and candidate measurements took one to six GPU-hours per run. Even so, by the cluster's own accounting, our runs used slightly less GPU time than the external systems (Table 2). We ran the audits outside every system's budget. The comparison therefore gives our agents no extra compute, and part of their budget went to verification rather than search.

Table 3: *Outcomes of the kernel's checks in the short study, against a three-seed audit outside the kernel.*

Check	Quantity	Rate
Measurement	credits confirmed by the audit	118/120 (98%)
Review	credits held as drafts	68/120 (57%)
Sandbox	cells that ran network commands	26/259 (10%)

Table 4: *Cost per cell of our agents: medians over 263 short-study cells and 10 main-study cells.*

Quantity per cell	Short	Main
GPU-hours (metered)	4.7	48.9
Evaluations launched by the agent	12	220
Kernel measurements	1	7
Sessions (wake-ups)	8	64
Model input / output tokens (M)	4.1 / 0.11	54.7 / 1.10
Model cost (USD)	2.53	57.36
GPU cost (USD)	10.35	107.73

5.4 Controlled comparisons

Author model. With the harness fixed and six GPU-hours per cell, audited gains vary widely across author models (Figure 5b). GLM 5.2 earns a credited gain in nearly all of its cells. Two other open models earn smaller gains, and three earn none. The author model therefore decides whether the kernel credits any gain at all.

One agent or a team. Larger compute budgets yield larger audited gains (Figure 5a). At the sizes we tried, one agent with a given budget and a team sharing the same total reach similar scores. Most small-budget cells stop at the learning-rate-tuning plateau. Cells pass it in one of two ways, by trying a different optimizer or by stacking a second idea on a tuned schedule. Teams pass it mostly by trying more different ideas, and single agents more often by stacking. The teams' agents read each other's activity list, but no agent sent a message to another, and switching communication on changed neither the teams' final changes nor their scores (Section 7).

5.5 The guarantees in practice

Across the 120 credited results of the short study (Table 3), the kernel credited on average slightly less than the audit confirmed, so its single paired measurement errs on the conservative side. All 120 credited results also improved on the audit's fresh seeds, and 118 cleared the noise floor there, so the audit confirmed nearly every credit. Review answers a different question, whether a measured gain comes from an acceptable change. Judges kept 68 of the 120 credited results as drafts, most often because a change bundled several edits or lacked an ablation. The audits confirmed the gains of most of these drafts, so this strictness is a cost of the design. A repository's maintainers can lower it by relaxing the judges' instructions. Some agents also ran network commands from their shell, mostly `git fetch` against the private repository, which failed without credentials. These attempts look like routine engineering rather than tampering.

The kernel's measurements took about a fifth of the short study's GPU-hours (Table 4), because each

Table 5: *Design properties of autonomous-research systems (license in parentheses), checked against code or papers, October 2026.* ✓ *yes,* ◐ *partial,* ✗ *no,* – *not documented.* Harness*: drives existing coding agents.* Indep. verify*: someone other than the author re-measures a claim before crediting it.* Isolation*: sandboxed, with credentials and evaluator out of reach.* Multi-agent*: independent agents on one repository, with messaging.* Async*: agents launch experiments, suspend, and resume.* GPU acct.*: GPU-hour metering.*

System	Harness	Indep. verify	Isolation	Multi-agent	Async	GPU acct.
autoresearch (no license)	✓	✗	✗	✗	✗	✗
AIDE (MIT)	✗	✗	✗	✗	✗	✗
aira-dojo (CC BY-NC)	✗	◐	✓	✗	✗	✗
R&D-Agent (MIT)	◐	◐	◐	✗	✗	✗
OpenEvolve (Apache-2.0)	◐	✗	✗	✗	✗	✗
ShinkaEvolve (Apache-2.0)	◐	✗	◐	✗	◐	✗
llm-speedrunner (CC BY-NC)	◐	✗	✗	✗	✗	✗
AI Scientist v2 (custom)	✗	◐	✗	✗	✗	✗
CodeScientist (Apache-2.0)	✗	◐	◐	✗	✗	✗
AlphaEvolve (closed)	✗	◐	–	✗	◐	✗
Outerloop (Apache-2.0)	✓	✓	✓	✓	✓	✓

submission runs twice and revised submissions are measured again. Model calls cost less than GPU time, because sessions sleep while experiments run and a cell resumes its session only a handful of times. The guarantees are therefore paid for mainly in GPU time.

5.6 Noise floor and number of seeds

How often does the kernel credit a change with no effect, and how often does it credit a real gain? The noise floor and the number of seeds set these two rates, FPR and Recall (Section 3). We estimate both from the audits' runs of the unmodified code (Figure 6a). Two different programs run on the same seed are nearly uncorrelated ($\rho \approx 0.06$), so we treat the two sides of a measurement as independent draws from these runs. With one seed and our 64-step floor, about one in seven changes with no effect would be credited (Figure 6b). Measuring on three seeds lowers this to 1.8%, but it triples the cost of each measurement. Large real gains are almost always credited (Figure 6d). A 64-step gain, equal to the floor, is credited only slightly more than half the time, whatever the number of seeds, because its average measurement sits right at the floor (Figure 6c). In practice, the three-seed audit confirmed almost every credited result in our study (Table 3). One seed is therefore a reasonable low-cost default, and maintainers who need a lower false-positive rate can pay for more seeds.

6 Related work

Autoresearch and self-improvement. Autonomous research systems now number in the dozens, and many release their code. Some edit an existing codebase and keep whatever improves a metric, such as Karpathy's autoresearch (Karpathy, 2026). The tree-search agents developed on MLE-bench work the same way, including AIDE (Jiang et al., 2025), aira-dojo (Toledo et al., 2025), ML-Master (Liu et al., 2025b)

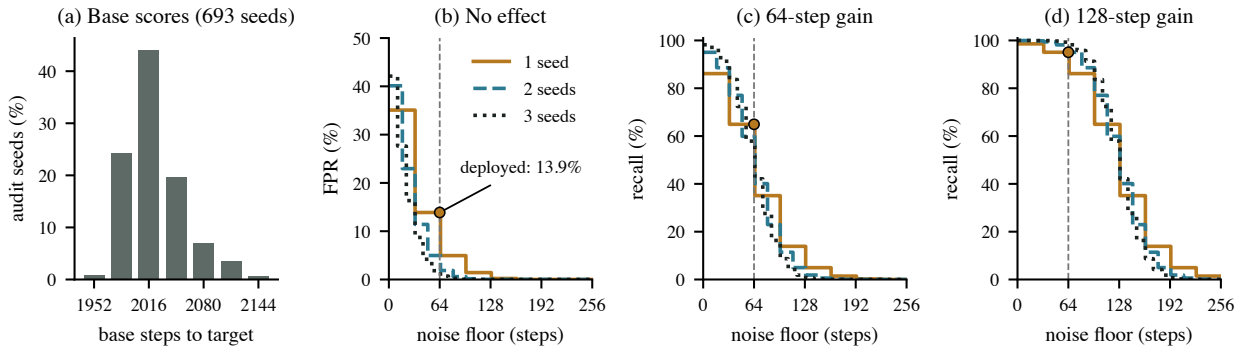


Figure 6: *Seed noise and credit rates. (a) Steps to target of the unmodified code on 693 audit seeds. (b–d) Probability of credit by noise floor, with one, two or three seeds per side (two, four or six runs), computed exactly from independent draws of (a): (b) a change with no effect, (c) a 64-step gain, (d) a 128-step gain. Dashed line and dot: our setting.*

and R&D-Agent (Yang et al., 2025). The evolutionary line in this group runs from FunSearch (Romera-Paredes et al., 2024) through AlphaEvolve (Novikov et al., 2025) to OpenEvolve (Sharma, 2025) and ShinkaEvolve (Lange et al., 2026). Others go from an idea to a manuscript, as in the AI Scientist (Lu et al., 2024; Yamada et al., 2025), Agent Laboratory (Schmidgall et al., 2025), CodeScientist (Jansen et al., 2025), DeepScientist (Weng et al., 2026), Curie (Kon et al., 2025) and Kosmos (Mitchener et al., 2025). ResearchAgent generates ideas without running them (Baek et al., 2025). A related line lets agents improve their own code, from a scaffold that improves the program that improves programs (Zelikman et al., 2023) to coding agents that evolve by editing themselves (Zhang et al., 2025a). These systems differ mainly in their search strategy. Table 5 compares them instead on the properties Outerloop is built around. In most of them, the system is both the author and the evaluator of its changes. Runs are bounded by step counts or model spend rather than GPU-hours. Multi-agent structure means parallel workers inside one search or a pipeline of roles. In Outerloop, we leave the search to the agents and provide the structure around it, with independent measurement, isolation, GPU budgets, and several independent authors on one repository.

Agent evaluation: audits and benchmarks. Benchmarks evaluate research agents on fixed tasks, including RE-Bench (Wijk et al., 2025), MLE-bench (Chan et al., 2025), MLAgentBench (Huang et al., 2024), MLGym (Nathani et al., 2025), MLRC-Bench (Zhang et al., 2025b), EXP-Bench (Kon et al., 2026), PaperBench (Starace et al., 2025) and the Automated LLM Speedrunning benchmark (Zhao et al., 2025). AstaBench compares research agents under controlled cost and tools (Bragg et al., 2026). Their graders sit outside the agent, for example on a private test split or in a fresh container. A recent checklist shows that weak graders can misstate agent results by large margins (Zhu et al., 2025). Audits of open-ended systems find the failures that such graders guard against. METR recorded reward hacking in 30.4% of 128 RE-Bench runs, including overwriting the timer and monkey-patching the evaluator (METR, 2025). Audits of the AI Scientist and Agent Laboratory found metric substitution, selection on test scores and fabricated results (Luo et al., 2025; Beel et al., 2025). Outside agent research, MLPerf fixes a quality target, requires repeated runs, and reviews submissions (Mattson et al., 2020). We apply the same rules to every claim during the search, and our fresh-seed audits follow the practice of these benchmarks.

Agent infrastructure. Coding-agent platforms give agents a purpose-built interface and a sandbox (Yang et al., 2024; Wang et al., 2025). Multi-agent frameworks assign agents roles that pass messages (Wu et al., 2024; Hong et al., 2024; Qian et al., 2024; Li et al., 2023). AIOS schedules and isolates many agents with

an operating-system kernel (Mei et al., 2025). These systems treat agents as cooperating parts of one application. Among research systems, native cluster orchestration is rare, with ShinkaEvolve, aira-dojo and llm-speedrunner as exceptions. We instead treat each agent as an untrusted program on a shared cluster, with metered experiments and claims measured before they count.

7 Discussion and limitations

Independent measurement. The kernel closes potential routes for result inflation in an autoresearch loop. Measurement on held-out seeds removes the luck of a chosen seed, and the agent's sandbox and edit scope keep the agent away from the evaluation code. The baseline autoresearch systems, which evaluate their own solutions, over-claimed by varying amounts in both short and long searches, whereas the kernel's measurements agreed with the fresh-seed audit within noise. Fresh-seed measurement addresses the main source of over-claim in our study. Outerloop is more than a re-evaluation step, however, and its separation of roles and edit scopes may also guard against other forms of result tampering. Over-claim also grows with the number of candidates a system evaluates, as ShinkaEvolve's rising over-claim shows (Section 5). We therefore expect self-evaluation to inflate results more as autoresearch moves to longer and wider searches, unless measurement is done outside the search. Independent measurement has a cost, however, as verification took about a fifth of the study's GPU-hours. Measuring on more seeds lowers the false-positive rate, but it costs more GPU-hours. A higher noise floor also lowers it without extra compute, but it misses small real gains.

Taste and novelty. The kernel decides whether a gain is real, and the judge agents decide whether the change behind it is acceptable for the repository. Neither decides whether the idea reflects good research taste or is new. A repository's maintainers can at least steer taste through a guidance file, like the one our agents received in the main study. We treat taste as a configurable layer, separate from the kernel's guarantees. Novelty is a different matter, because the benchmark does not distinguish a recalled technique from a new one. In our study, much of the progress on the benchmark came from recalling the Muon optimizer, and in the main study the systems differed mainly in whether and when they adopted it. MLGym shows a similar pattern: frontier agents improve mostly by tuning hyperparameters (Nathani et al., 2025). Human review does not close this gap easily either, since ideas that reviewers rate highly can score lower once they are run (Si et al., 2026). Our comparisons therefore measure the recall of known techniques and say little about the discovery of new ones. A judge with web search could compare an idea against published work. We turned web search off in our study to keep the runs reproducible, but a deployment can turn it on at any time.

Coordination. Our agents did not use the channels to divide work (Section 5), which echoes the coordination failures reported for other multi-agent systems (Cemri et al., 2025). The teammates' activity list showed current experiments but not their results, which may have encouraged copying rather than division of labor. Sharing measured results between teammates, or instructing agents to divide the work, would change only what the agents receive, not the kernel's guarantees. Because the kernel only delivers messages and leaves their use to the agents, this result reflects how current agents use the channels.

Beyond computational experiments. Our experiments are computational, but the same structure can extend to other scientific domains. To the kernel, an experiment is a job that it launches and charges against a budget. The same structure could govern agents that run physical experiments in automated laboratories (Boiko et al., 2023). The edit scope would be the experimental protocol, and the budget would be instrument time or materials. The independent measurement would be a replicate on held-out samples,

and the judges would review the protocol for validity and safety. Replicates are slow and costly, however, so independent measurement takes a larger share of the budget. The kernel must also trust the instrument's readings, because it cannot check them the way it re-runs a training script. False discoveries are especially costly in physical science, which makes independent measurement valuable there, and connecting the kernel to automated laboratories is a natural next step.

Limitations. Our experiments use one small training-speed benchmark and mostly one author model. The benchmark's noise is large relative to the gains, which strengthens selection bias. We have not measured the over-claim on benchmarks with less noise. To run the external systems on our cluster, we wrapped each one in an adapter that sends its evaluations out as cluster jobs and, for AIDE, lengthens timeouts and adds retries. These changes may have affected the systems' behavior. AIDE's runs often died on model-call errors, so its score at 48 GPU-hours rests on two runs. In our study, shell commands could still reach the network, and the attempts we observed failed for lack of credentials (Section 5). By design, the sandbox withholds credentials but does not filter network traffic, because separating legitimate downloads such as packages or datasets from harmful traffic would need a policy for each repository. Finally, the kernel guarantees only measurement and edit scope. A change that games the evaluation without leaving the editable paths is left to the judge agents, which may not catch it.

8 Conclusion

Research agents can now run experiments for days, which makes the trustworthiness of their results critical. We present Outerloop, a kernel that gives these agents a fixed outer structure in which it measures their claims, guards the evaluation code, and meters their compute. Inside this structure, the agents decide what to try and how to work. On a training-speed benchmark, our fresh-seed audit shows that the self-evaluating baseline systems over-claim their gains, while the kernel's measurements match the audit at a modest cost in compute. With measurement and budget held fixed, we also use the kernel as an instrument for comparing agents, and find that much of the current progress comes from recalling known techniques. Outerloop is open source, and we will release the audited results. We hope they make longer and larger agent searches on real repositories both safe to run and possible to trust.

Acknowledgments

We thank Chris Hoang, Hui Dai, and Ying Wang for software feedback, bug fixes, and comments on the paper. The study ran on cloud credits from the NYU Google Cloud Platform (GCP) for Research Grant Program, under the project "Autoresearch: Open, Verified AI Research Agents." This research was supported by the DOE Genesis Mission, Visko Platform, Pionex, a Google TPU Award, a Google ML and Systems Junior Faculty Award, and the Institute of Information & Communications Technology Planning & Evaluation (IITP) with a grant funded by the Ministry of Science and ICT (MSIT) of the Republic of Korea in connection with the Global AI Frontier Lab International Collaborative Research Agreement (No. RS-2024-00469482 & RS-2024-00509279). The compute is supported by the NYU High Performance Computing resources, services, and staff expertise. We gratefully acknowledge use of the research computing resources of the Empire AI Consortium, Inc, with support from Empire State Development of the State of New York, the Simons Foundation, and the Secunda Family Foundation.

References

Baek, J., Jauhar, S. K., Cucerzan, S., and Hwang, S. J. (2025). Researchagent: Iterative research idea generation over scientific literature with large language models. In *Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*.

Baker, B., Huizinga, J., Gao, L., Dou, Z., Guan, M. Y., Madry, A., Zaremba, W., Pachocki, J., and Farhi, D. (2025). Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. arXiv preprint arXiv:2503.11926.

Beel, J., Kan, M.-Y., and Baumgart, M. (2025). Evaluating Sakana's AI scientist: Bold claims, mixed results, and a promising future?

Boiko, D. A., MacKnight, R., Kline, B., and Gomes, G. (2023). Autonomous chemical research with large language models. *Nature*, 624(7992):570–578.

Bouthillier, X., Delaunay, P., Bronzi, M., Trofimov, A., Nichyporuk, B., Szeto, J., Sepah, N., Raff, E., Madan, K., Voleti, V., Kahou, S. E., Michalski, V., Serdyuk, D., Arbel, T., Pal, C., Varoquaux, G., and Vincent, P. (2021). Accounting for variance in machine learning benchmarks. In *Proceedings of Machine Learning and Systems (MLSys)*.

Bragg, J., D'Arcy, M., Balepur, N., Bareket, D., Dalvi, B., Feldman, S., Haddad, D., Hwang, J. D., Jansen, P., Kishore, V., Majumder, B. P., Naik, A., Rahamimov, S., Richardson, K., Singh, A., Surana, H., Tiktinsky, A., Vasu, R., Wiener, G., Anastasiades, C., Candra, S., Dunkelberger, J., Emery, D., Evans, R., Hamada, M., Huff, R., Kinney, R., Latzke, M., Lochner, J., Lozano-Aguilera, R., Nguyen, C., Rao, S., Tanaka, A., Vlahos, B., Clark, P., Downey, D., Goldberg, Y., Sabharwal, A., and Weld, D. S. (2026). Astabench: Rigorous benchmarking of ai agents with a scientific research suite. In *International Conference on Learning Representations (ICLR)*.

Cawley, G. C. and Talbot, N. L. C. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107.

Cemri, M., Pan, M. Z., Yang, S., Agrawal, L. A., Chopra, B., Tiwari, R., Keutzer, K., Parameswaran, A., Klein, D., Ramchandran, K., Zaharia, M., Gonzalez, J. E., and Stoica, I. (2025). Why do multi-agent llm systems fail? In *Advances in Neural Information Processing Systems (NeurIPS)*.

Chan, J. S., Chowdhury, N., Jaffe, O., Aung, J., Sherburn, D., Mays, E., Starace, G., Liu, K., Maksin, L., Patwardhan, T., Weng, L., and Madry, A. (2025). MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations (ICLR)*.

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., and Schmidhuber, J. (2024). Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*.

Huang, Q., Vora, J., Liang, P., and Leskovec, J. (2024). Mlagentbench: Evaluating language agents on machine learning experimentation. In *International Conference on Machine Learning (ICML)*.

Jansen, P., Tafjord, O., Radensky, M., Siangliulue, P., Hope, T., Mishra, B. D., Majumder, B. P., Weld, D. S., and Clark, P. (2025). Codescientist: End-to-end semi-automated scientific discovery with code-based experimentation. In *Findings of the Association for Computational Linguistics: ACL 2025*.

Jiang, Z., Schmidt, D., Srikanth, D., Xu, D., Kaplan, I., Jacenko, D., and Wu, Y. (2025). AIDE: AI-driven exploration in the space of code.

Jordan, K., Bernstein, J., Rappazzo, B., @fernbear.bsky.social, Vlado, B., Jiacheng, Y., Cesista, F., Koszarsky, B., and @Grad62304977 (2024a). modded-nanogpt: Speedrunning the NanoGPT baseline. GitHub repository.

Jordan, K., Jin, Y., Boza, V., You, J., Cesista, F., Newhouse, L., and Bernstein, J. (2024b). Muon: An optimizer for hidden layers in neural networks. Blog post.

Karpathy, A. (2026). autoresearch. https://github.com/karpathy/autoresearch.

Kon, P. T. J., Liu, J., Ding, Q., Qiu, Y., Yang, Z., Huang, Y., Srinivasa, J., Lee, M., Chowdhury, M., and Chen, A. (2025). Curie: Toward rigorous and automated scientific experimentation with ai agents. arXiv preprint arXiv:2502.16069.

Kon, P. T. J., Liu, J., Zhu, X., Ding, Q., Peng, J., Xing, J., Huang, Y., Qiu, Y., Srinivasa, J., Lee, M., Chowdhury, M., Zaharia, M., and Chen, A. (2026). Exp-bench: Can ai conduct ai research experiments? In *International Conference on Learning Representations (ICLR)*.

Lange, R. T., Imajuku, Y., and Cetin, E. (2026). ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations (ICLR)*.

Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., and Ghanem, B. (2023). Camel: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Liu, J., Su, J., Yao, X., Jiang, Z., Lai, G., Du, Y., Qin, Y., Xu, W., Lu, E., Yan, J., Chen, Y., Zheng, H., Liu, Y., Liu, S., Yin, B., He, W., Zhu, H., Wang, Y., Wang, J., Dong, M., Zhang, Z., Kang, Y., Zhang, H., Xu, X., Zhang, Y., Wu, Y., Zhou, X., and Yang, Z. (2025a). Muon is scalable for llm training. arXiv preprint arXiv:2502.16982.

Liu, Z., Cai, Y., Zhu, X., Zheng, Y., Chen, R., Wen, Y., Wang, Y., E, W., and Chen, S. (2025b). Ml-master: Towards ai-for-ai via integration of exploration and reasoning. arXiv preprint arXiv:2506.16499.

Lu, C., Lu, C., Lange, R. T., Foerster, J., Clune, J., and Ha, D. (2024). The AI scientist: Towards fully automated open-ended scientific discovery.

Luo, Z., Kasirzadeh, A., and Shah, N. B. (2025). The more you automate, the less you see: Hidden pitfalls of AI scientist systems. In *NeurIPS 2025 Workshop on AI for Science (AI4Science)*.

Mattson, P., Cheng, C., Coleman, C., Diamos, G., Micikevicius, P., Patterson, D., Tang, H., Wei, G.-Y., Bailis, P., Bittorf, V., Brooks, D., Chen, D., Dutta, D., Gupta, U., Hazelwood, K., Hock, A., Huang, X., Ike, A., Jia, B., Kang, D., Kanter, D., Kumar, N., Liao, J., Ma, G., Narayanan, D., Oguntebi, T., Pekhimenko, G., Pentecost, L., Reddi, V. J., Robie, T., John, T. S., Tabaru, T., Wu, C.-J., Xu, L., Yamazaki, M., Young, C., and Zaharia, M. (2020). Mlperf training benchmark. In *Proceedings of Machine Learning and Systems (MLSys)*.

Mei, K., Zhu, X., Xu, W., Hua, W., Jin, M., Li, Z., Xu, S., Ye, R., Ge, Y., and Zhang, Y. (2025). Aios: Llm agent operating system. In *Conference on Language Modeling (COLM)*.

METR (2025). Recent frontier models are reward hacking. https://metr.org/blog/2025-06-05-recent-reward-hacking/.

Mitchener, L., Yiu, A., Chang, B., Bourdenx, M., Nadolski, T., Sulovari, A., Landsness, E. C., Barabasi, D. L., Narayanan, S., Evans, N., Reddy, S., Foiani, M., Kamal, A., Shriver, L. P., Cao, F., Wassie, A. T., Laurent, J. M., Melville-Green, E., Caldas, M., Bou, A., Roberts, K. F., Zagorac, S., Orr, T. C., Orr, M. E., Zwezdaryk, K. J., Ghareeb, A. E., McCoy, L., Gomes, B., Ashley, E. A., Duff, K. E., Buonassisi, T., Rainforth, T., Bateman, R. J., Skarlinski, M., Rodriques, S. G., Hinks, M. M., and White, A. D. (2025). Kosmos: An ai scientist for autonomous discovery. arXiv preprint arXiv:2511.02824.

Nathani, D., Madaan, L., Roberts, N., Bashlykov, N., Menon, A., Moens, V., Budhiraja, A., Magka, D., Vorotilov, V., Chaurasia, G., Hupkes, D., Cabral, R. S., Shavrina, T., Foerster, J., Bachrach, Y., Wang, W. Y., and Raileanu, R. (2025). Mlgym: A new framework and benchmark for advancing ai research agents. In *Conference on Language Modeling (COLM)*.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J. R., Mehrabian, A., Kumar, M. P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., and Balog, M. (2025). AlphaEvolve: A coding agent for scientific and algorithmic discovery.

Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., and Sun, M. (2024). Chatdev: Communicative agents for software development. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J. S., Wang, P., Fawzi, O., Kohli, P., and Fawzi, A. (2024). Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475.

Saltzer, J. H. and Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308.

Schmidgall, S., Su, Y., Wang, Z., Sun, X., Wu, J., Yu, X., Liu, J., Moor, M., Liu, Z., and Barsoum, E. (2025). Agent laboratory: Using llm agents as research assistants. In *Findings of the Association for Computational Linguistics: EMNLP 2025*.

Sharma, A. (2025). OpenEvolve: An open-source evolutionary coding agent. GitHub repository.

Si, C., Hashimoto, T., and Yang, D. (2026). The ideation-execution gap: Execution outcomes of llm-generated versus human research ideas. In *International Conference on Learning Representations (ICLR)*.

Singh, S., Nan, Y., Wang, A., D'Souza, D., Kapoor, S., Üstün, A., Koyejo, S., Deng, Y., Longpre, S., Smith, N. A., Ermis, B., Fadaee, M., and Hooker, S. (2025). The leaderboard illusion. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Starace, G., Jaffe, O., Sherburn, D., Aung, J., Chan, J. S., Maksin, L., Dias, R., Mays, E., Kinsella, B., Thompson, W., Heidecke, J., Glaese, A., and Patwardhan, T. (2025). Paperbench: Evaluating ai's ability to replicate ai research. In *International Conference on Machine Learning (ICML)*.

Toledo, E., Hambardzumyan, K., Josifoski, M., Hazra, R., Baldwin, N., Audran-Reiss, A., Kuchnik, M., Magka, D., Jiang, M., Lupidi, A. M., Lupu, A., Raileanu, R., Niu, K., Shavrina, T., Gagnon-Audet, J.-C., Shvartsman, M., Sodhani, S., Miller, A. H., Charnalia, A., Dunfield, D., Wu, C.-J., Stenetorp, P., Cancedda, N., Foerster, J. N., and Bachrach, Y. (2025). Ai research agents for machine learning: Search, exploration, and generalization in mle-bench. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., Tran, H. H., Li, F., Ma, R., Zheng, M., Qian, B., Shao, Y., Muennighoff, N., Zhang, Y., Hui, B., Lin, J., Brennan, R., Peng, H., Ji, H., and Neubig, G. (2025). Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*.

Weng, Y., Zhu, M., Xie, Q., Sun, Q., Lin, Z., Liu, S., and Zhang, Y. (2026). Deepscientist: Advancing frontier-pushing scientific findings progressively. In *International Conference on Learning Representations (ICLR)*.

Wijk, H., Lin, T., Becker, J., Jawhar, S., Parikh, N., Broadley, T., Chan, L., Chen, M., Clymer, J., Dhyani, J., Ericheva, E., Garcia, K., Goodrich, B., Jurkovic, N., Karnofsky, H., Kinniment, M., Lajko, A., Nix, S., Sato, L., Saunders, W., Taran, M., West, B., and Barnes, E. (2025). RE-bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. In *International Conference on Machine Learning (ICML)*.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. (2024). Autogen: Enabling next-gen llm applications via multi-agent conversation. In *Conference on Language Modeling (COLM)*.

Yamada, Y., Lange, R. T., Lu, C., Hu, S., Lu, C., Foerster, J., Clune, J., and Ha, D. (2025). The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. (2024). Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Yang, X., Yang, X., Fang, S., Zhang, Y., Wang, J., Xian, B., Li, Q., Li, J., Xu, M., Li, Y., Pan, H., Zhang, Y., Liu, W., Shen, Y., Chen, W., and Bian, J. (2025). R&D-Agent: An llm-agent framework towards autonomous data science. arXiv preprint arXiv:2505.14738.

Yoo, A. B., Jette, M. A., and Grondona, M. (2003). SLURM: Simple Linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing (JSSPP)*, volume 2862 of *Lecture Notes in Computer Science*, pages 44–60. Springer.

Zelikman, E., Lorch, E., Mackey, L., and Kalai, A. T. (2023). Self-taught optimizer (STOP): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*.

Zhang, J., Hu, S., Lu, C., Lange, R., and Clune, J. (2025a). Darwin Godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*.

Zhang, Y., Khalifa, M., Bhushan, S., Murphy, G. D., Logeswaran, L., Kim, J., Lee, M., Lee, H., and Wang, L. (2025b). Mlrc-bench: Can language agents solve machine learning research challenges? In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.

Zhao, B., Magka, D., Jiang, M., Li, X., Raileanu, R., Shavrina, T., Gagnon-Audet, J.-C., Niu, K., Sodhani, S., Shvartsman, M., Lupu, A., Lupidi, A., Toledo, E., Hambardzumyan, K., Josifoski, M., Foster, T., Cipolina-Kun, L., Charnalia, A., Dunfield, D., Miller, A. H., Mac Aodha, O., Foerster, J., and Bachrach, Y. (2025). The automated LLM speedrunning benchmark: Reproducing NanoGPT improvements. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.

Zhong, Z., Raghunathan, A., and Carlini, N. (2026). Impossiblebench: Measuring llms' propensity of exploiting test cases. In *International Conference on Learning Representations (ICLR)*.

Zhu, Y., Jin, T., Pruksachatkun, Y., Zhang, A., Liu, S., Cui, S., Kapoor, S., Longpre, S., Meng, K., Weiss, R., Barez, F., Gupta, R., Dhamala, J., Merizian, J., Giulianelli, M., Coppock, H., Ududec, C., Sekhon, J., Steinhardt, J., Kellermann, A., Schwettmann, S., Zaharia, M., Stoica, I., Liang, P., and Kang, D. (2025). Establishing best practices for building rigorous agentic benchmarks. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.