SCIENCEIDE: SCALING SCIENTIFIC EXPERIENCE THROUGH EXECUTABLE ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models learned to converse from text and to act through executable code; discovery-oriented intelligence must learn through experimentation. Science concentrates the capabilities this next stage demands: forming hypotheses, intervening in complex systems, and revising beliefs against evidence. Yet scientific software and expertise do not readily become experience that agents can learn from: repositories depend on fragile toolchains and unwritten conventions, and correctness depends on scientific behavior and numerical tolerances. This is the *scientific experience bottleneck*. We introduce **ScienceIDE**, infrastructure that makes expert knowledge reusable for agent development: domain experts define scientific cases and acceptance criteria, and AI agents help turn these decisions into executable environments. Because the checks are fixed before any task is written, task factories generate an open-ended supply of repair, implementation, and acceleration tasks whose difficulty is raised by design, each validated through execution and scientific verification. The same environments produce graded trajectories for supervised fine-tuning, online rewards for reinforcement learning, and evidence for model evaluation. The loop runs in both directions: science supplies verifiable experience for developing agents, and agents return to science, where their interactions generate new tasks. ScienceIDE is therefore not only about using AI to accelerate science; it is about using science to develop more capable AI.

1 INTRODUCTION

Language-model capability grows with the experience available for learning. *Chat intelligence* learns from human knowledge and instruction following (Ouyang et al., 2022; Christiano et al., 2017). *Coding intelligence* adds repositories, compilers, and tests: actions become executable, consequences observable, and failures reusable for learning (Jimenez et al., 2023; Pan et al., 2025; Jain et al., 2025; Yang et al., 2025). Such experience enables reinforcement learning with verifiable rewards (DeepSeek-AI et al., 2025; Wei et al., 2025). *Discovery intelligence* requires a further step: identifying worthwhile questions, comparing hypotheses through interventions, and revising judgments against external evidence (Silver & Sutton, 2025; Sutton & Barto, 2018). Science concentrates these demands in sustained interactions with code, data, and simulators. It is a testbed and training ground for developing intelligence through experimentation.

This progression exposes the *scientific experience bottleneck*. Scientific repositories depend on heterogeneous toolchains, fragile configurations, and unwritten conventions. Correctness depends on physical observables, numerical tolerances, and scientific behavior. Before runnable code can support learning, it needs meaningful tasks, behavioral validation, graded feedback, and recorded interactions. Papers, repositories, and datasets do not automatically provide this experience.

ScienceIDE turns scientific expertise into reusable infrastructure for agent development (Figure 1). Domain experts define module boundaries, scientific cases, and acceptance criteria; agents help package these decisions into executable environments. Checks are fixed before task authoring. Environment-specific factories then generate candidates whose validity is established through execution, and a shared episode interface supplies evaluation evidence, supervised fine-tuning (SFT) trajectories, and reinforcement-learning (RL) rewards. Models and trainers can change while the

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

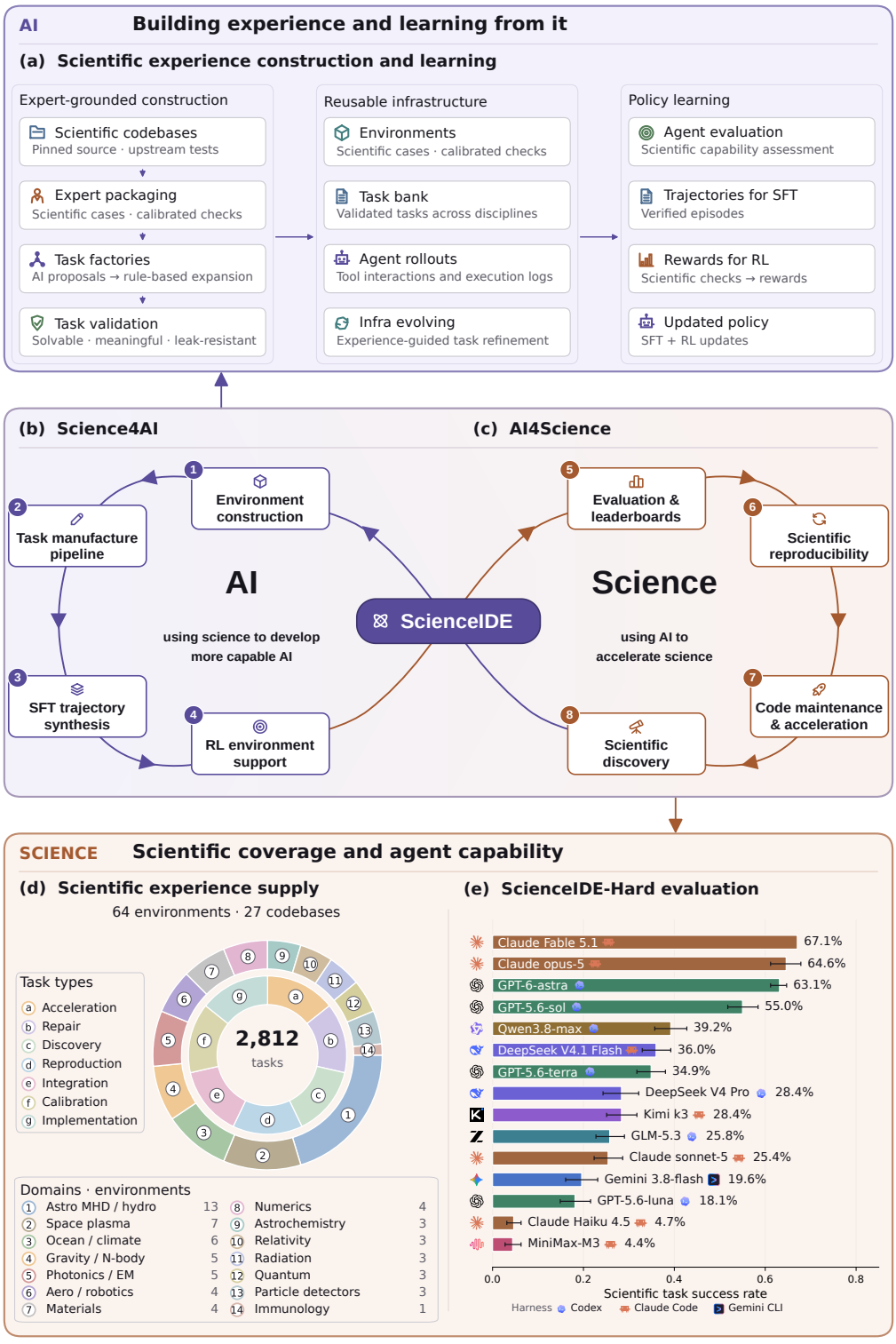


Figure 1: **ScienceIDE connects scientific work with agent learning.** (a) Expert-grounded construction provides reusable infrastructure for agent evaluation and learning. (b) Science4AI turns scientific experience into more capable agents. (c) AI4Science applies agents to scientific work and discovery. (d) Environment coverage by domain (outer ring) and task taxonomy (equal inner sectors, not task frequencies). (e) Model–harness performance on ScienceIDE-Hard; icons identify harnesses, and intervals summarize repeated runs.

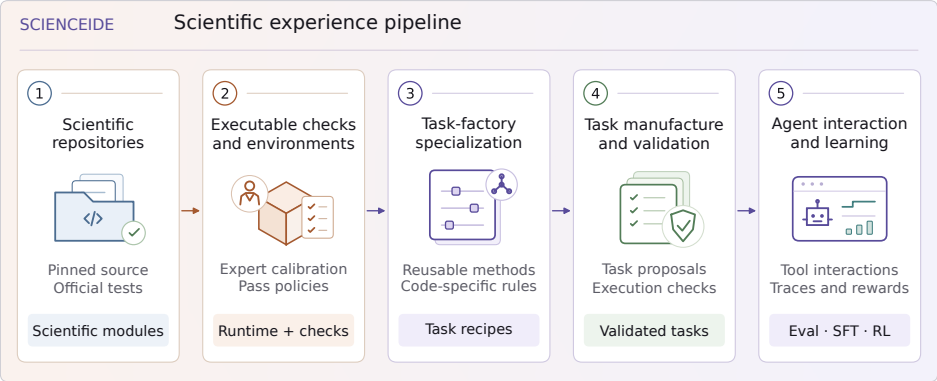


Figure 2: **ScienceIDE technical overview.** A versioned repository is packaged with a runtime and scientific checks. Environment-specific factories propose candidates; validation admits tasks whose episodes support evaluation, SFT, and RL.

scientific contract remains reusable. Agent interactions can also propose new scientific objectives, which re-enter validation before becoming tasks.

The contribution is the connection between expert curation, repeated task production, and model learning. The frozen registry contains 64 environments from 27 codebases and 2,812 tasks; environments are scientific modules, not independent repositories. We evaluate scientific execution on ScienceIDE-Hard and study public-benchmark SFT transfer and hinted, within-environment RL. Repair and implementation provide a verifiable starting point toward discovery-oriented work. The broader aim is a sustained exchange: science supplies grounded experience for developing agents, and better agents extend the scientific work from which future experience is built.

2 Related Work

Scientific models and agents support prediction, computation, and discovery, from protein structure and weather forecasting to research workflows and program search over algorithms (Jumper et al., 2021; Lam et al., 2022; Lu et al., 2024; Gottweis et al., 2025; Novikov et al., 2025; Romera-Paredes et al., 2024), with validation ranging from proof checkers to wet-lab experiments (Ghareeb et al., 2025; Mitchener et al., 2025). Scientific evaluations include coding (Tian et al., 2024), workflow execution (Chen et al., 2025), reproduction (Siegel et al., 2024; Starace et al., 2025), and repository-level software work (Duston et al., 2025; Xu et al., 2026); expert authoring cost, saturation, and public-answer contamination motivate reusable task generation with scientific verification beyond successful compilation. Closest to our substrate, AInsteinBench and SWE-bench Science source maintainer pull requests from scientific repositories and show how software tests under-specify scientific contracts; both provide evaluation instances rather than a mechanism for producing verified tasks at scale.

Coding shows how executable tasks supply training experience: SWE-bench established repository evaluation (Jimenez et al., 2023), SWE-Gym, R2E-Gym, and SWE-smith provide executable training tasks at scale (Pan et al., 2025; Jain et al., 2025; Yang et al., 2025), and verifiable rewards drive reasoning and agent training (DeepSeek-AI et al., 2025; Wei et al., 2025). Scientific interaction environments expose simulator control or broader agent actions (Lagemann et al., 2026; Bhan et al., 2024; Nathani et al., 2025; Narayanan et al., 2024; Jansen et al., 2024), but scaling such experience across scientific codebases must account for simulation cost, numerical variation, and multiple valid outcomes: a verifier has to name which quantities are compared, on which workloads and tolerances, and how partial progress becomes reward. ScienceIDE instead makes an expert-approved scientific module and its checks a reusable unit for producing tasks that feed evaluation, SFT, and RL (comparisons in Appendix A).

3 METHOD

3.1 SCIENCEIDE: REUSABLE SCIENTIFIC EXPERIENCE

ScienceIDE integrates environments, tasks, verification, and model learning (Figure 2). Its construction unit is a scientific *module* within a versioned *codebase*. A module owns a coherent scientific responsibility and executable coverage. An *environment* packages an approved module with its runtime and checks. A *factory* specializes authoring procedures to that environment; its validated *tasks* specify an initial workspace, deliverable, and verifier. An *episode* is an agent interaction that produces artifacts, a trajectory, and measured outcomes. Multiple environments can share a codebase or toolchain while retaining distinct scientific responsibilities and source provenance.

Experts retain decisions about observables, equivalence, and acceptance. Agents assist reconstruction and authoring, while deterministic execution establishes whether candidates meet the contract. This separation amortizes scientific judgment across task families. The integrated loop also supports maintenance, acceleration, reproduction, tool integration, and proposals for new objectives. The expanded construction stages and system views are in Appendices A and B.

3.2 EXECUTABLE SCIENTIFIC ENVIRONMENTS

Construction starts from a pinned revision of production scientific software. An agent inspects dependencies, license, and build assumptions, builds the source, and runs official tests and examples. These runs expose output formats, numerical variation, expensive paths, and execution hazards. The agent proposes modules with explicit inputs, outputs, algorithm stages, owned implementation paths, excluded responsibilities, and supporting tests. A domain expert reviews the decomposition and coverage. The approved module is packaged with an editable workspace, experiment adapter, scientific checks, and a separate private verifier; provenance and observed hazards remain attached to the runtime.

The registry snapshot (2026-09-12) provides 64 environments derived from 27 scientific codebases and 2,812 manufactured tasks: 2,515 repair, 295 implementation, and two acceleration tasks. Each environment designates the workload an acceleration task must speed up. A companion collection provides 1,076 executable checks of numerical outputs and physical invariants. The task-supply, scientific-contract, and agent-evaluation cohorts have different denominators (Appendix C.2); task count does not establish independent scientific coverage. The complete source and license inventory is in Appendix J.

Acceleration illustrates why the contract matters: accelerator ports remain hand-written and vendor-dependent, with double precision a default rather than a derived requirement (Rossazza et al., 2025; An et al., 2025; Romeo et al., 2026; Düben & Palmer, 2014; Düben et al., 2015); fixing observables and tolerances lets repair, reconstruction, and acceleration share one acceptance contract.

3.3 SCIENTIFIC CHECKS, TASK FACTORIES, AND VALIDITY

Checks precede tasks. Official unit tests, regression tests, and example problems are traced to a module's responsibility. Each case is retained as a check, excluded with a reason, or recorded as a gap; a custom check requires curator agreement. A check fixes inputs, graded outputs, and a *pass policy*. Examples without a shipped reference use the pinned build for comparison, anchored by published values, convergence, or conserved quantities.

A pointwise policy compares graded values using $|c - r| \leq a + \rho|r|$, with exact equality when $a = \rho = 0$. Outputs are aligned by their scientific identity, such as particle ID, before comparison. Storage order, timings, adaptive-step counts, rank layout, random draws, and eigenvector phase are not observables. When no pointwise bound contains legitimate variation while rejecting a real fault, an invariants policy compares moments, distributions, conserved quantities, or integral norms. The graded window is shortened first if it retains the relevant physics.

Nominal and variant initial conditions, each run independently and required to reproduce its reference at full reward, reveal numerical sensitivity, and an optional alternative legitimate build measures cross-build separation; these are finite calibration evidence, not automatic tolerance rules. The curator and domain expert finalize observable, policy, tolerance, window, and variant from source

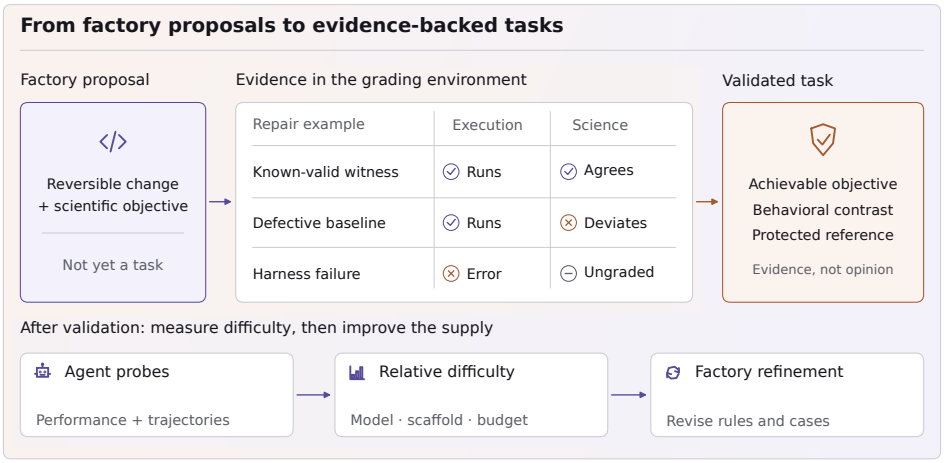


Figure 3: **Scientific task validity requires executable evidence.** For injected repairs, the known-valid witness and defective baseline both run, but only the witness satisfies the scientific checks. Harness failures remain ungraded. Design probes guide proposals; validated tasks receive model-relative difficulty measurements that inform factory refinement.

mechanisms, and each check carries a plain-language warrant for which scientific bias its bound distinguishes (Appendix C).

Factories reuse the contract. Shared procedures handle reversible edits, execution, and artifact assembly; environment-specific rules identify meaningful transformations and active paths. The seven authoring categories are Acceleration, Repair, Discovery, Reproduction, Integration, Calibration, and Implementation. Repair and Implementation automate candidate expansion through mutation and excision; the remaining categories expose interfaces for expert-specified objectives, data, procedures, and resource constraints. Acceleration adds a resource criterion after correctness; reproduction grades a runnable procedure's outputs.

Validity requires executable evidence. Proposed transformations, including syntactic mutations (Just et al., 2014b;a), are tested in the final grading environment. For injected repair, a known-valid witness must pass, the unfixed baseline must leave headroom, and the reference repair must remove the induced check failure. Compilation, native execution, and coverage precede admission where applicable. Silent mutations may be explicit controls; ambiguous specifications, unreachable branches, and harness failures remain invalid or ungraded. Packaging screens answer leakage and records execution-integrity failures. Other objectives require an attainable target, observable outcome, and informative credit for alternatives (Figure 3).

Scientific checks expose partial accomplishment. With scientific agreement r and unfixed-build score f, repair reward is

$$r_{\text{repair}} = \max\left(0, \frac{r - f}{1 - f}\right), \qquad 0 \le f < 1. \tag{1}$$

Task records distinguish scientific disagreement from infrastructure failure and preserve reasons for audit. Task-family admission labels and design probes do not substitute for measured difficulty under a named model, harness, budget, and date.

3.4 Learning from Scientific Experience

A shared episode interface. Agents inspect code and inputs, make changes, run experiments, and submit artifacts to private checks. The harness records actions, observations, rewards, execution status, and resource use. Evaluation holds tasks and budgets fixed; SFT supervises selected agent actions, including tool calls; RL connects policy-controlled rollouts to verifier rewards. Tasks can be selected by environment, domain, family, or measured difficulty without rebuilding scientific content.

Supervised fine-tuning. We fine-tune Qwen3.5-4B, Qwen3.5-9B, and Qwen2.5-72B-Instruct on GPT-5.6-sol demonstrations selected by numerical-equivalence verification. The corpus has 4,567 training segments from 564 tasks and 544 validation segments from 81 tasks, with no shared task identifier. New assistant actions and tool calls are supervised; instructions, observations, reused context, and flagged repetitive or failed actions are masked. Training uses ms-swift and LoRA on all linear layers for three epochs. Final checkpoints are evaluated without selection on public-benchmark performance. Related-variant overlap was not independently audited. Construction and hyperparameters are in Appendix H.1.

Reinforcement learning. RL independently starts from base Qwen3.5-4B, without SFT initialization. LAPS, a 3D pseudo-spectral Hall-MHD solver (Shi et al., 2024), and MITgcm-biogeo supply 99 and 87 repair tasks over three and nine graded scientific cases. L1 hints identify the defect's file, line, and edit class without its repair. The verifier compiles modified code, runs scientific simulations, and supplies outcome-only reward in $[0, 1]$. Generation uses vLLM (Kwon et al., 2023); PSRL (Li et al., 2026; Zhou et al., 2026), a customization of veRL (Sheng et al., 2024), drives a black-box harness. Concurrent rollout and training use disjoint GPUs, with token-level truncated importance sampling correcting one-step policy staleness.

For $G = 8$ trajectories with rewards R_i, the group-relative advantage is

$$\hat{A}_i = R_i - \frac{1}{G}\sum_{j=1}^{G} R_j. \tag{2}$$

Following Dr. GRPO (Liu et al., 2025), it omits standard-deviation normalization. PPO-style clipping uses $\varepsilon_{\text{low}} = 0.2$ and $\varepsilon_{\text{high}} = 0.3$ (Yu et al., 2025), with token-mean aggregation. Budget-truncated trajectories retain their reward in the group baseline but are masked out of the token loss. A harness cutoff does not establish that the preceding actions were unhelpful; penalizing all their tokens can encourage shorter responses instead of scientific progress. The reward-side length penalty is also disabled. The full objective, termination mask, resource allocation, and run-level comparison are in Appendix H.2.

4 Experiments and results

4.1 Experimental setup

ScienceIDE-Hard contains 85 validated tasks from 18 environments derived from PLUTO, Athena++, MITgcm, LAPS, and PHANTOM: 52 repair and 33 implementation tasks. They cover astrophysical flows, plasma physics, ocean and atmospheric modeling, and particle simulations, probing multi-site edits and convention-faithful reconstruction in codebases of 10^4 to 10^5 lines. Selection combines reference-solver difficulty probing with execution of the reference solution and defective baseline.

Fifteen models from eight providers run through Codex, Claude Code, or Gemini CLI with the same task-specific containers and instructions, no localization hints, and a one-hour episode budget. Strict scientific success requires $r_{\text{repair}} = 1$; incomplete or budget-exhausted deliveries fail. Results compare model–harness systems, weight tasks equally, average within-task repeats, and describe execution variability on the fixed test set. Fourteen models have three valid attempts on every task; Fable is singly measured. The protocol, retrieval corrections, and accounting boundaries are in Appendix D.

4.2 Scientific agent performance

Fable 5.1 has the highest observed success at 67.1%, followed by Opus 5 at 64.6% and Astra at 63.1%; Sol reaches 55.0%, and the other eleven agents score below 40% (Figure 1e). Even leading systems leave roughly one third of tasks unsolved. The top point estimates do not establish a statistical ordering: Fable is singly measured and the Opus and Astra intervals overlap.

Budget and resource use. At ten minutes, Astra reaches 49.6% success versus Fable's 25.9%; Fable overtakes it at approximately 31 minutes. Figure 4 shows retrospective completion profiles, not

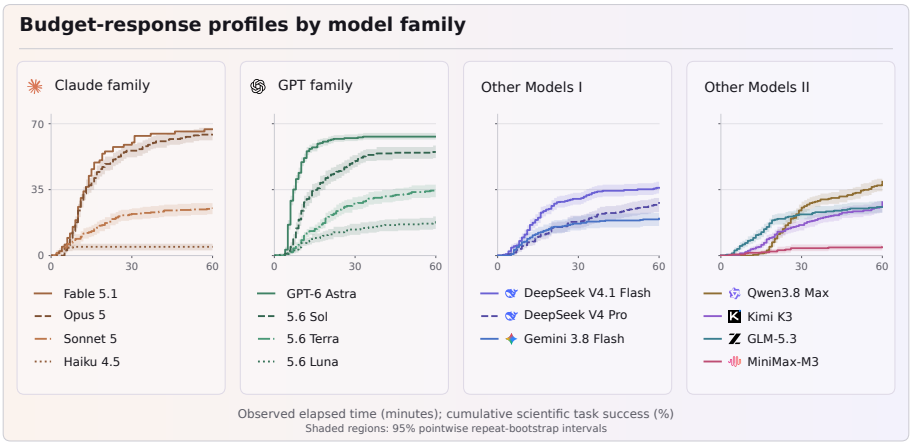


Figure 4: **Budget changes the observed ordering.** Cumulative scientific task success over the recorded one-hour episodes for all fifteen agents. Shading denotes pointwise 95% within-task repeat-bootstrap intervals where available. These are retrospective completion profiles, not reruns under shorter assigned budgets.

reruns with shorter budgets. Fable's 67.1% success costs an estimated $7.90 per task, versus Astra's 63.1% at $3.56; costs use archived rates or recorded harness estimates. Runtime and output volume have descriptive Spearman correlations with success of -0.22 and 0.01. Scientific correctness and resource use are distinct dimensions; full curves and effort comparisons appear in Appendix E.

Failure and scientific conventions. A review of Fable and Astra covers 358 attempts, including 125 unsuccessful attempts, on all 85 tasks. Reference-convention mismatches account for 71.4% and 64.9% of their respective task-balanced failures; wrong-target edits account for 10.7% and 17.0%. Runnable reconstructions can miss timestep-retention rules, coefficient precision, or state conventions, and passing comparisons between repeated candidate runs need not test the assigned defect. The model-assisted annotations describe supported mechanisms, not isolated causes; 17 attempts remain unresolved. Full failure, repeat-instability, coverage, and trajectory diagnostics are preserved in Appendix F.

4.3 SFT TRANSFER TO PUBLIC BENCHMARKS

Initial and SFT checkpoints are evaluated on identical items within each model pair, covering scientific knowledge, mathematical reasoning, code understanding, repair, and generation. Metrics are reported separately, without an overall average. Figure 5 retains 15 model–benchmark comparisons with gains of at least 3 percentage points in screening. For 4B, HumanEvalFix JavaScript gains 10.98 points; QuixBugs Java rises from 0.400 to 0.500 (+10.0 points), and CodeXGLUE defect detection from 0.422 to 0.492 (+7.03 points), still below 0.5. For 9B, BBH Word Sorting rises from 0.240 to 0.576 (+33.6 points).

Disjoint confirmation items test selected findings (Table 3 in Appendix H.1). These include a decline on HumanEvalFix Python; the 72B HumanEvalFix C++ screening gain also does not persist on confirmation. Intervals are unadjusted for multiple comparisons. The findings support selected code, reasoning, and knowledge gains from verified scientific demonstrations, without establishing uniform improvement or transfer to unseen scientific codebases. All screening comparisons, confirmation details, and teacher-forced validation diagnostics remain in Appendices G and H.1.

4.4 RL ON HELD-OUT SCIENTIFIC TASKS

With budget-truncated trajectories masked from the loss, mean held-out verifier reward after 30 training steps rises from 0.357 to 0.857 ($2.4\times$) on LAPS and from 0.286 to 0.571 ($2.0\times$) on MITgcm-biogeo (Figure 6). The comparison uses 14 LAPS and 21 MITgcm-biogeo tasks under matched harnesses and L1 hints. LAPS has an 85/14 source split; MITgcm-biogeo records 64/23,

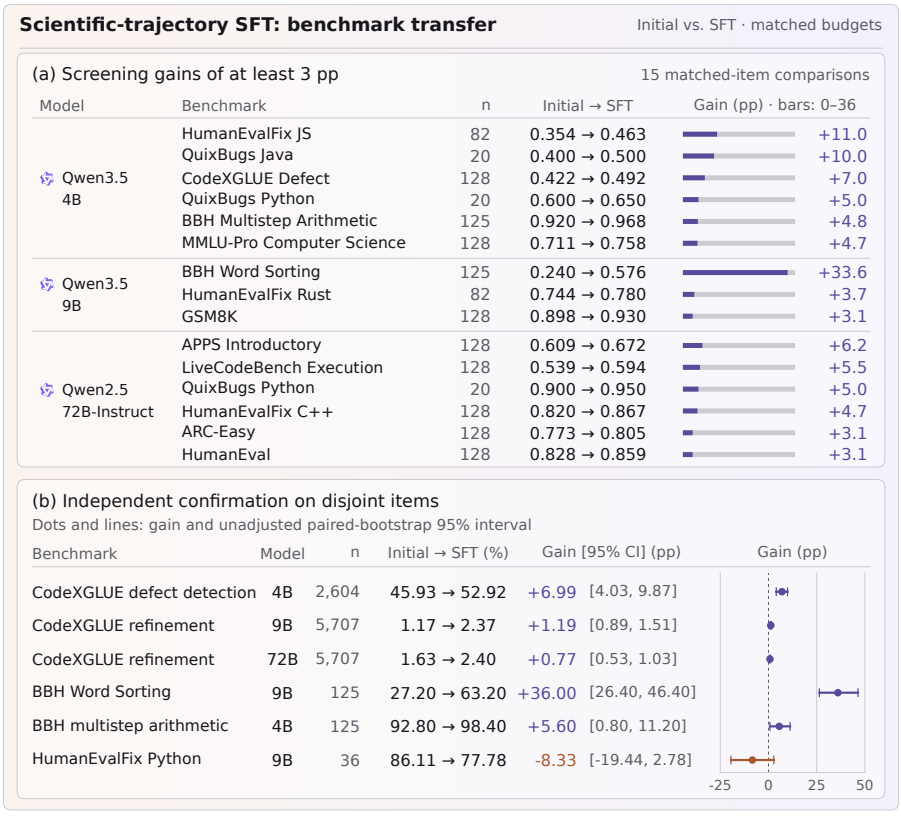


Figure 5: **Benchmark improvements.** (a) The 15 model–benchmark comparisons with screening gains of at least 3 percentage points, grouped by model. Bars show the gain; each row reports the measured initial → SFT scores on a 0–1 scale and the item count. Differences are computed before rounding the displayed scores. (b) Confirmation comparisons on disjoint items with unadjusted paired-bootstrap 95% intervals for the gain in percentage points, including one decline (Appendix H.1).

with 21 of the 23 validation tasks in the logged evaluation. The rebuilt 66/21 split is not substituted for the reported run.

Comparing the first and last five recorded steps, mean training reward rises from 0.427 to 0.828 on LAPS and 0.381 to 0.597 on MITgcm-biogeo; budget truncation falls from 39.5% to 6.6% and from 34.1% to 23.8%. Tokens per turn increase from 1103 to 1135 and from 780 to 914, respectively. In the unmasked comparison, reward rises from 0.339 to 0.573 before collapsing to 0.078, while tokens per turn fall from 1125 to 327. The run-level evidence motivates separating budget cutoffs from failed repairs in policy optimization. These are hinted, within-environment comparisons without between-seed variance estimates; they do not establish transfer across codebases.

5 Discussion and Limitations

The evidence concerns reference-verifiable software tasks, predominantly repair and implementation in computational physics and geoscience. Open-ended discovery, broader task-family coverage, and the 1,000-environment release remain development targets. Generated variants can share code paths and assumptions; the hard set probes horizon and scale only up to a one-hour budget and two coupled edit sites; measured difficulty stays conditional on solver, budget, and available localization channels (Appendix I).

Verification is a scientific modeling choice. Agreement with selected observables and tolerances does not establish correctness outside those checks, and legitimate alternatives may disagree with

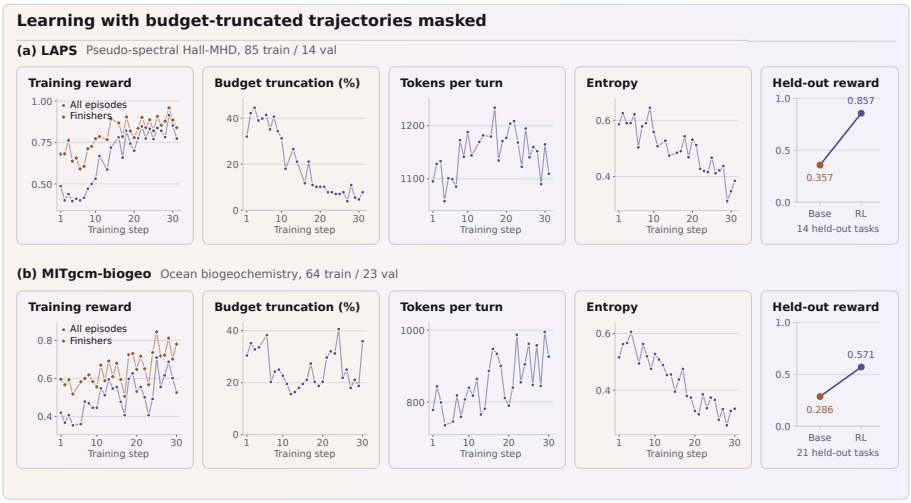


Figure 6: **Training dynamics with budget-truncated trajectories masked, and held-out reward.** Every recorded step of both runs; headers give source bank splits (the logged MITgcm-biogeo validation set has 21 of its 23 tasks, Appendix H.2). Right column: held-out reward of base Qwen3.5-4B and its step-30 checkpoint on 14 LAPS and 21 MITgcm-biogeo tasks under matched harnesses and hints; values are mean shaped verifier rewards.

the reference. Independent external audits and adversarial reward-hacking evaluations remain outstanding, and the verification cost an agent faces in-container is not yet varied systematically across tasks. Factories amortize expert curation; expansion still depends on expertise and upstream redistribution rights. Changes to repositories, dependencies, or hardware require revalidation, which the execution-based gates make mechanical but not free.

Scores compare model–harness systems; serving speed and unequal repeat coverage limit ranking. Retrieval controls cannot exclude pretrained knowledge of public code. Legacy images retain timestamp cues, and difficulty selection predates complete retrieval isolation, so selection is not shown to be contamination-free. SFT partitioning is verified by task identifier rather than related variants; its gains concern selected public benchmarks. RL gains concern hinted tasks within training environments and do not estimate between-seed variance. Neither study establishes transfer to unseen codebases.

6 Conclusion

ScienceIDE turns scientific expertise into reusable environments, checks, and learning experience. Its experiments expose gaps between plausible code and verified scientific execution, alongside selected public-benchmark SFT gains and hinted, within-environment RL gains. Fixed scientific contracts connect expert curation to repeated task production without replacing scientific judgment. Progress from Chat through Coding toward Discovery Intelligence will require broader scientific coverage, transfer across codebases and task families, and checks that accommodate competing hypotheses and novel outcomes. ScienceIDE provides infrastructure for that exchange between scientific work and the development of more capable agents, and the same contracts are meant to admit new task families, from acceleration and porting to falsifiable scientific claims, as the environment set grows.

AI USE STATEMENT

We used generative AI tools to polish the writing of this paper, including grammar, wording, and condensing the text to fit the page limit. The model-assisted components of our method, environment construction and trajectory review, are part of the research itself and are described in Section 3.2 and Appendices B and F.4. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The reported studies involve software environments and model-generated interactions, with no human subjects. Upstream scientific code retains its recorded licenses; source-specific license information appears in Appendix J, and redistribution remains subject to those licenses. Scientific verification applies to the specified checks and tolerances and does not establish correctness beyond that scope.

REPRODUCIBILITY STATEMENT

The paper records pinned scientific sources, check calibration, validation anchors, evaluation budgets, repeat estimators, and training settings in Appendices B, C, D, and H.2. SFT preprocessing and confirmation protocols are in Appendix H.1. Supporting code and source records are designated for anonymous supplementary material. Historical retrieval corrections, incomplete repeat coverage, and the RL split provenance are documented with the corresponding results.

REFERENCES

Jelle Aalbers, Joran R. Angevaare, Dacheng Xu, Daniel Wenz, Chris Tunnell, et al. AxFoundation/strax: v2.2.3. Zenodo software release, July 2026. URL https://doi.org/10.5281/zenodo.21262220. Upstream generic streaming-analysis framework and signal-processing kernels used by all xenon-stream-processing checks.

Marwa Abdulhai, Isadora White, Charlie Snell, Charles Sun, Joey Hong, Yuexiang Zhai, et al. Lmrl gym: Benchmarks for multi-turn reinforcement learning with language models. *arXiv preprint arXiv:2311.18232*, 2023.

Yifu An, Yuxi Chen, Hongyang Zhou, Alexander Gaenko, and Gábor Tóth. BATSRUS GPU: Faster-than-real-time magnetospheric simulations with a block-adaptive grid code. *The Astrophysical Journal*, 981:188, 2025. doi: 10.3847/1538-4357/adaf15.

Anthropic. System card: Claude Opus 4 & claude Sonnet 4. technical report, 2025.

Anthropic. Formalizing fermat's last theorem, 2026. https://www.anthropic.com/research/formalizing-fermats-last-theorem.

T D Arber, K Bennett, C S Brady, A Lawrence-Douglas, M G Ramsay, N J Sircombe, P Gillies, R G Evans, H Schmitz, A R Bell, and C P Ridgers. Contemporary particle-in-cell approach to laser-plasma modelling. *Plasma Physics and Controlled Fusion*, 57(11):113001, 2015. doi: 10.1088/0741-3335/57/11/113001. URL https://doi.org/10.1088/0741-3335/57/11/113001.

Nathan Bell, Luke N. Olson, Jacob Schroder, and Ben Southworth. PyAMG: Algebraic multigrid solvers in Python. *Journal of Open Source Software*, 8(87):5495, 2023. doi: 10.21105/joss.05495. URL https://doi.org/10.21105/joss.05495.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *arXiv preprint arXiv:1207.4708*, 2012.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *arXiv preprint arXiv:2106.04399*, 2021.

Luke Bhan, Yuexin Bian, Miroslav Krstic, and Yuanyuan Shi. PDE control gym: A benchmark for data-driven boundary control of partial differential equations. In *Learning for Dynamics and Control (L4DC), PMLR v242*, pp. 1083–1095, 2024. URL https://proceedings.mlr.press/v242/bhan24a.html.

Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Pangu-Weather: A 3d high-resolution model for fast and accurate global weather forecast. *arXiv preprint arXiv:2211.02556*, 2022.

Daniil A. Boiko, Robert MacKnight, and Gabe Gomes. Emergent autonomous scientific research capabilities of large language models. *arXiv preprint arXiv:2304.05332*, 2023.

Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, et al. TorchRL: A data-driven decision-making library for PyTorch. *arXiv preprint arXiv:2306.00577*, 2023.

Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, et al. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

Sébastien Bubeck, Christian Coester, Ronen Eldan, Timothy Gowers, Yin Tat Lee, Alexandru Lupsasca, Mehtaab Sawhney, Robert Scherrer, Mark Sellke, Brian K. Spears, Derya Unutmaz, Kevin Weil, Steven Yin, and Nikita Zhivotovskiy. Early science acceleration experiments with GPT-5. *arXiv preprint arXiv:2511.16072*, 2025.

Shiyi Cao, Sumanth Hegde, Dacheng Li, Tyler Griggs, Shu Liu, Eric Tang, et al. SkyRL-v0: Train real-world long-horizon agents via reinforcement learning. technical report, 2025.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, et al. MLE-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Mouxiang Chen, Lei Zhang, et al. SWE-Universe: Scale real-world verifiable environments to millions. *arXiv preprint arXiv:2602.02361*, 2026.

Ziru Chen, Shijie Chen, Yuting Ning, et al. ScienceAgentBench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *International Conference on Learning Representations (ICLR)*, pp. 96934–96990, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/f12b4df26344f3be803c06b555252efe-Abstract-Conference.html.

Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*, 2017.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, et al. Textworld: A learning environment for text-based games. *arXiv preprint arXiv:1806.11532*, 2018.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, et al. DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602:414–419, 2022.

Andrea Del Prete, Nicolas Mansard, Oscar E. Ramos, Olivier Stasse, and Francesco Nori. Implementing torque control with high-ratio gear boxes and without joint-torque sensors. *International Journal of Humanoid Robotics*, 13(1):1550044, 2016. doi: 10.1142/S0219843615500449. URL https://doi.org/10.1142/S0219843615500449.

Haonan Duan, Stephen Zhewen Lu, Caitlin Fiona Harrigan, et al. Measuring scientific capabilities of language models with a systems biology dry lab. *arXiv preprint arXiv:2507.02083*, 2025.

Peter D. Düben and T. N. Palmer. Benchmark tests for numerical weather forecasts on inexact hardware. *Monthly Weather Review*, 142(10):3809–3829, 2014. doi: 10.1175/MWR-D-14-00110.1.

Peter D. Düben, Francis P. Russell, Xinyu Niu, Wayne Luk, and T. N. Palmer. On the use of programmable hardware and reduced numerical precision in earth-system modeling. *Journal of Advances in Modeling Earth Systems*, 7(3):1393–1408, 2015. doi: 10.1002/2015MS000494.

Titouan Duston, Shuo Xin, Yang Sun, Daoguang Zan, Aoyan Li, et al. AInsteinBench: Benchmarking coding agents on scientific repositories. *arXiv preprint arXiv:2512.21373*, 2025.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, et al. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *arXiv preprint arXiv:2206.08853*, 2022.

Alireza Ghafarollahi and Markus J. Buehler. Sciagents: Automating scientific discovery through multi-agent intelligent graph reasoning. *arXiv preprint arXiv:2409.05556*, 2024.

Ali Essam Ghareeb, Benjamin Chang, Ludovico Mitchener, Angela Yiu, Caralyn J. Szostkiewicz, Jon M. Laurent, et al. Robin: A multi-agent system for automating scientific discovery. *arXiv preprint arXiv:2505.13400*, 2025.

Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, July 2021. doi: 10.22331/q-2021-07-06-497. URL https://doi.org/10.22331/q-2021-07-06-497.

Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, et al. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *arXiv preprint arXiv:2411.04872*, 2024.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, et al. Accelerating scientific discovery with co-scientist. *arXiv preprint arXiv:2502.18864*, 2025.

Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. Textarena. *arXiv preprint arXiv:2504.11442*, 2025.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

Ammar Hakim and Gkeyll Team. Gkeyll: A multi-scale, multi-physics simulation framework. Source code, GitHub, 2026. Source snapshot c6be0d59f45c8194e54fda032aae6acb35e3fbb8, committed 2026-09-02; MIT license; documentation: https://gkeyll.readthedocs.io/en/latest/.

Lauri Himanen, Marc O. J. Jäger, Eiaki V. Morooka, Filippo Federici Canova, Yashasvi S. Ranawat, David Z. Gao, Patrick Rinke, and Adam S. Foster. DScribe: Library of descriptors for machine learning in materials science. *Computer Physics Communications*, 247:106949, 2020. doi: 10.1016/j.cpc.2019.106949. URL https://doi.org/10.1016/j.cpc.2019.106949.

Bin Hu, Marco Raveri, Noemi Frusciante, and Alessandra Silvestri. Effective field theory of cosmic acceleration: an implementation in CAMB. *Physical Review D*, 89:103530, 2014. doi: 10.1103/PhysRevD.89.103530. URL https://doi.org/10.1103/PhysRevD.89.103530.

Jian Hu, Xibin Wu, Wei Shen, Jason Klein Liu, Zilin Zhu, Weixun Wang, et al. OpenRLHF: An easy-to-use, scalable and high-performance RLHF framework. *arXiv preprint arXiv:2405.11143*, 2024.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

Yuchen Huang, Sijia Li, et al. Environment scaling for interactive agentic experience collection: A survey. *arXiv preprint arXiv:2511.09586*, 2025.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environments and hybrid verifiers for scaling open-weights SWE agents. *arXiv preprint arXiv:2504.07164*, 2025.

Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, et al. DISCOVERYWORLD: A virtual environment for developing and evaluating automated scientific discovery agents. *arXiv preprint arXiv:2406.06769*, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, et al. SWE-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596:583–589, 2021.

René Just, Darioush Jalali, and Michael D. Ernst. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *International Symposium on Software Testing and Analysis (ISSTA)*, pp. 437–440, 2014a. doi: 10.1145/2610384.2628055. URL https://doi.org/10.1145/2610384.2628055.

René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? In *FSE 2014: Proceedings of the ACM SIGSOFT 22nd Symposium on the Foundations of Software Engineering*, pp. 654–665, Hong Kong, 2014b. URL https://homes.cs.washington.edu/~mernst/pubs/mutation-effectiveness-fse2014-abstract.html.

M. H. Kelsey, R. Agnese, Y. F. Alam, I. Ataee Langroudy, E. Azadbakht, D. Brandt, R. Bunker, B. Cabrera, Y.-Y. Chang, H. Coombes, R. M. Cormier, M. D. Diamond, E. R. Edwards, E. Figueroa-Feliciano, J. Gao, P. M. Harrington, Z. Hong, M. Hui, N. A. Kurinsky, R. E. Lawrence, B. Loer, M. G. Masten, E. Michaud, E. Michielin, J. Miller, V. Novati, N. S. Oblath, J. L. Orrell, W. L. Perry, P. Redl, T. Reynolds, T. Saab, B. Sadoulet, K. Serniak, J. Singh, Z. Speaks, C. Stanford, J. R. Stevens, J. Strube, D. Toback, J. N. Ullom, B. A. VanDevender, M. R. Vissers, M. J. Wilson, J. S. Wilson, B. Zatschler, and S. Zatschler. G4CMP: Condensed matter physics simulation using the Geant4 toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 1055: 168473, 2023. doi: 10.1016/j.nima.2023.168473. URL https://doi.org/10.1016/j.nima.2023.168473.

Patrick W. Kenneally, Scott Piggott, and Hanspeter Schaub. Basilisk: A flexible, scalable and modular astrodynamics simulation framework. *Journal of Aerospace Information Systems*, 17(9): 496–507, 2020. doi: 10.2514/1.I010762. URL https://doi.org/10.2514/1.I010762.

Patrick Tser Jern Kon, Jiachen Liu, Qiuyi Ding, Yiming Qiu, Zhenning Yang, Yibo Huang, et al. Curie: Toward rigorous and automated scientific experimentation with ai agents. *arXiv preprint arXiv:2502.16069*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Symposium on Operating Systems Principles (SOSP)*, 2023. URL https://arxiv.org/abs/2309.06180.

Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, et al. The nethack learning environment. *arXiv preprint arXiv:2006.13760*, 2020.

Christian Lagemann et al. The HydroGym reinforcement learning platform for fluid dynamics. *Nature*, 657:369–376, 2026. doi: 10.1038/s41586-026-10917-6. arXiv:2512.17534.

Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, et al. GraphCast: Learning skillful medium-range global weather forecasting. *arXiv preprint arXiv:2212.12794*, 2022.

Neill Lambert, Eric Giguère, Paul Menczel, Boxi Li, Patrick Hopf, Gerardo Suárez, Marc Gali, Jake Lishman, Rushiraj Gadhvi, Rochisha Agarwal, Asier Galicia, Nathan Shammah, Paul D. Nation, J. R. Johansson, Shahnawaz Ahmed, Simon Cross, Alexander Pitchford, and Franco Nori. QuTiP 5: The quantum toolbox in Python. *Physics Reports*, 1153:1–62, 2026. doi: 10.1016/j.physrep.2025.10.001. URL https://doi.org/10.1016/j.physrep.2025.10.001.

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, et al. OpenSpiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*, 2019.

Jon M. Laurent, Joseph D. Janizek, Michael Ruzo, Michaela M. Hinks, Michael J. Hammerling, Siddharth Narayanan, et al. Lab-bench: Measuring capabilities of language models for biology research. *arXiv preprint arXiv:2407.10362*, 2024.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, et al. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*, 2022.

Haoyang Li, Sheng Lin, Fangcheng Fu, Yuming Zhou, Xiaodong Ji, Yanfeng Zhao, Lefeng Wang, Jie Jiang, and Bin Cui. Staleflow: Staleness-aware data management for mitigating data skewness in fully disaggregated rl post-training, 2026.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, et al. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, et al. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Victor Liu and Shanhui Fan. S^4: A free electromagnetic solver for layered periodic structures. *Computer Physics Communications*, 183(10):2233–2244, 2012. doi: 10.1016/j.cpc.2012.04.026. URL https://doi.org/10.1016/j.cpc.2012.04.026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Zichen Liu, Changyu Chen, Xinyi Wan, Chao Du, Wee Sun Lee, and Min Lin. Oat: A research-friendly framework for LLM online alignment. Software repository, 2024.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Zichen Liu, Anya Sims, Keyu Duan, Changyu Chen, Simon Yu, Xiangxin Zhou, et al. GEM: A gym for generalist LLMs. In *International Conference on Learning Representations (ICLR)*, pp. 110681–110700, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/b3956812189001f673d29c626bc897ae-Abstract-Conference.html.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, et al. Discoverybench: Towards data-driven discovery with large language models. *arXiv preprint arXiv:2407.01725*, 2024.

Benjamin W. L. Margolis and Kenneth R. Lyons. SimuPy flight vehicle toolkit. *Journal of Open Source Software*, 7(75):4299, 2022. doi: 10.21105/joss.04299. URL https://doi.org/10.21105/joss.04299.

John Marshall, Alistair Adcroft, Chris Hill, Lev Perelman, and Curt Heisey. A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers. *Journal of Geophysical Research: Oceans*, 102(C3):5753–5766, 1997. doi: 10.1029/96JC02775. URL https://doi.org/10.1029/96JC02775.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023.

A. Mignone, G. Bodo, S. Massaglia, T. Matsakos, O. Tesileanu, C. Zanni, and A. Ferrari. PLUTO: A numerical code for computational astrophysics. *The Astrophysical Journal Supplement Series*, 170(1):228–242, 2007. doi: 10.1086/513316. URL https://doi.org/10.1086/513316.

Ludovico Mitchener, Angela Yiu, Benjamin Chang, Mathieu Bourdenx, Tyler Nadolski, Arvis Sulovari, et al. Kosmos: An ai scientist for autonomous discovery. *arXiv preprint arXiv:2511.02824*, 2025.

Ansh Nagda, Prabhakar Raghavan, and Abhradeep Thakurta. Reinforced generation of combinatorial structures: Ramsey numbers. *arXiv preprint arXiv:2603.09172*, 2026. URL https://arxiv.org/abs/2603.09172.

Siddharth Narayanan, James D. Braza, Ryan-Rhys Griffiths, Manu Ponnapati, Albert Bou, Jon Laurent, et al. Aviary: training language agents on challenging scientific tasks. *arXiv preprint arXiv:2412.21154*, 2024.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, et al. MLGym: A new framework and benchmark for advancing ai research agents. *arXiv preprint arXiv:2502.14499*, 2025.

Alexander Novikov, Ngân V u, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Shyue Ping Ong, William Davidson Richards, Anubhav Jain, Geoffroy Hautier, Michael Kocher, Shreyas Cholia, Dan Gunter, Vincent L. Chevrier, Kristin A. Persson, and Gerbrand Ceder. Python Materials Genomics (pymatgen): A robust, open-source python library for materials analysis. *Computational Materials Science*, 68:314–319, 2013. doi: 10.1016/j.commatsci.2012.10.028. URL https://doi.org/10.1016/j.commatsci.2012.10.028.

OpenAI. GPT-5 system card. https://cdn.openai.com/gpt-5-system-card.pdf, 2025.

OpenAI. GPT-5.2 derives a new result in theoretical physics, 2026. https://openai.com/index/new-result-theoretical-physics/.

OpenAI, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, et al. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Ardavan F. Oskooi, David Roundy, Mihai Ibanescu, Peter Bermel, J. D. Joannopoulos, and Steven G. Johnson. Meep: A flexible free-software package for electromagnetic simulations by the FDTD method. *Computer Physics Communications*, 181(3):687–702, 2010. doi: 10.1016/j.cpc.2009.11.008. URL https://doi.org/10.1016/j.cpc.2009.11.008.

Long Ouyang, Jeffrey Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022. doi: 10.52202/068431-2011. URL https://proceedings.neurips.cc/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47717–47737. PMLR, 2025. URL https://proceedings.mlr.press/v267/pan25g.html.

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, et al. FourCastNet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

Long Phan et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Mihir Prabhudesai, Aryan Satpathy, et al. Solving physics olympiad via reinforcement learning on physics simulators. *arXiv preprint arXiv:2604.11805*, 2026.

Daniel J. Price, James Wurster, Terrence S. Tricco, Chris Nixon, Stéven Toupin, Alex Pettitt, Conrad Chan, Daniel Mentiplay, Guillaume Laibe, Simon Glover, Clare Dobbs, Rebecca Nealon, David Liptai, Hauke Worpel, Clément Bonnerot, Giovanni Dipierro, Giulia Ballabio, Enrico Ragusa, Christoph Federrath, Roberto Iaconi, Thomas Reichardt, Duncan Forgan, Mark Hutchison, Thomas Constantino, Ben Ayliffe, Kieran Hirsh, and Giuseppe Lodato. Phantom: A Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics. *Publications of the Astronomical Society of Australia*, 35:e031, 2018. doi: 10.1017/pasa.2018.25. URL https://doi.org/10.1017/pasa.2018.25.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3505–3506, 2020. doi: 10.1145/3394486.3406703. URL https://doi.org/10.1145/3394486.3406703.

Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. OpenAI technical report, 2019.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, et al. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Jie Ren. EDKit.jl. Julia software, GitHub repository, 2026. Version 0.5.0, commit 538fce882ab73e3af447f4bc6a1704d290c88aba; MIT license.

Alessandro Romeo, Nitin Shukla, Stefano Truzzi, Alessio Suriano, and Andrea Mignone. On the limits of performance portability in directive-based GPU programming. *arXiv preprint arXiv:2606.12753*, 2026.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

Marco Rossazza, Andrea Mignone, Matteo Bugli, Stefano Truzzi, Lubomir Riha, Tomas Panoc, Ondrej Vysocky, Nitin Shukla, Alessandro Romeo, and Vittoria Berta. The PLUTO code on GPUs: A first look at Eulerian MHD methods. *arXiv preprint arXiv:2511.20337*, 2025.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, et al. Agent laboratory: Using llm agents as research assistants. *arXiv preprint arXiv:2501.04227*, 2025.

N. A. Schwadron, L. Townsend, K. Kozarev, M. A. Dayeh, F. Cucinotta, M. Desai, M. Golightly, D. Hassler, R. Hatcher, M.-Y. Kim, A. Posner, M. PourArsalan, H. E. Spence, and R. K. Squier. Earth-Moon-Mars Radiation Environment Module framework. *Space Weather*, 8(1), 2010. doi: 10.1029/2009SW000523. URL https://doi.org/10.1029/2009SW000523.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. *arXiv preprint arXiv:2409.19256*, 2024.

Chen Shi, Anna Tenerani, Antonio Franco Rappazzo, and Marco Velli. LAPS: An MPI-parallelized 3D pseudo-spectral Hall-MHD simulation code incorporating the expanding box model. *Frontiers in Astronomy and Space Sciences*, 11:1412905, 2024. doi: 10.3389/fspas.2024.1412905. URL https://www.frontiersin.org/journals/astronomy-and-space-sciences/articles/10.3389/fspas.2024.1412905/full.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Zachary S. Siegel, Sayash Kapoor, Nitya Nadgir, Benedikt Stroebl, and Arvind Narayanan. CORE-Bench: Fostering the credibility of published research through a computational reproducibility agent benchmark. *arXiv preprint arXiv:2409.11363*, 2024.

David Silver and Richard S. Sutton. Welcome to the era of experience, 2025. URL https://storage.googleapis.com/deepmind-media/Era-of-Experience%20/The%20Era%20of%20Experience%20Paper.pdf. Chapter preprint for *Designing an Intelligence*, MIT Press.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, et al. Paperbench: Evaluating ai's ability to replicate ai research. *arXiv preprint arXiv:2504.01848*, 2025.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, et al. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2505.24760*, 2025.

James M. Stone, Kengo Tomida, Christopher J. White, and Kyle G. Felker. The Athena++ adaptive mesh refinement framework: Design and magnetohydrodynamic solvers. *The Astrophysical Journal Supplement Series*, 249(1):4, 2020. doi: 10.3847/1538-4365/ab929b. URL https://doi.org/10.3847/1538-4365/ab929b.

Gregor Sturm, Tamas Szabo, Georgios Fotakis, Marlene Haider, Dietmar Rieder, Zlatko Trajanoski, and Francesca Finotello. Scirpy: a Scanpy extension for analyzing single-cell T-cell receptor-sequencing data. *Bioinformatics*, 36(18):4817–4818, 2020. doi: 10.1093/bioinformatics/btaa611. URL https://doi.org/10.1093/bioinformatics/btaa611.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, 2018.

M. Szydagis, N. Barry, K. Kazkaz, J. Mock, D. Stolp, M. Sweany, M. Tripathi, S. Uvarov, N. Walsh, and M. Woods. NEST: a comprehensive model for scintillation yield in liquid xenon. *Journal of Instrumentation*, 6(10):P10002, 2011. doi: 10.1088/1748-0221/6/10/P10002. URL https://doi.org/10.1088/1748-0221/6/10/P10002.

Chenmien Tan, Simon Yu, Lei Wei, Lanbo Lin, Ze Zhang, Yuanwu Xu, Chenhao Jiang, Tianyuan Yang, Sicong Xie, and Guannan Zhang. RL2: Ray less reinforcement learning. Software repository, 2025.

Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, et al. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*, 2022.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, et al. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

J. K. Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*, 2020.

Minyang Tian, Luyu Gao, Shizhuo D. Zhang, et al. SciCode: A research coding benchmark curated by scientists. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks*, volume 37, pp. 30624–30650, 2024. doi: 10.52202/079017-0963. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/36850592258c8c41cecdaa3dea5ff7de-Abstract-Datasets_and_Benchmarks_Track.html.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Silviu-Marian Udrescu and Max Tegmark. Ai feynman: a physics-inspired method for symbolic regression. *arXiv preprint arXiv:1905.11481*, 2019.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, et al. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Weixun Wang, Shaopan Xiong, Gengru Chen, Wei Gao, Sheng Guo, Yancheng He, et al. Reinforcement learning optimization for large-scale learning: An efficient and user-friendly scaling library. *arXiv preprint arXiv:2506.06122*, 2025.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, et al. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024a.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024b.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, et al. SWE-RL: Advancing llm reasoning via reinforcement learning on open software evolution. *arXiv preprint arXiv:2502.18449*, 2025.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, et al. Re-bench: Evaluating frontier ai r&d capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, et al. DeepSeek-Prover: Advancing theorem proving in llms through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*, 2024.

Zhipeng Xu, Jiahao Lu, Yining Zheng, Yuxin Wang, and Xipeng Qiu. SWE-bench Science: Can coding agents resolve engineering tasks in science? *arXiv preprint arXiv:2608.19799*, 2026.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, et al. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, et al. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, et al. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Xingdi Yuan, Morgane M. Moss, Charbel El Feghali, Chinmay Singh, Darya Moldavskaya, Drew MacPhee, et al. debug-gym: A text-based environment for interactive debugging. *arXiv preprint arXiv:2503.21557*, 2025.

Kevin Zakka. Mink: Python inverse kinematics based on MuJoCo. Software, August 2026. Version 1.3.0; benchmark source commit 14625beca2ce0918f88d1fc84a3c0cdb591e0729.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning. *arXiv preprint arXiv:2203.14465*, 2022.

Claudio Zeni, Robert Pinsler, Daniel Zügner, Andrew Fowler, Matthew Horton, Xiang Fu, et al. Mattergen: a generative model for inorganic materials design. *arXiv preprint arXiv:2312.03687*, 2023.

Xiangyuan Zhang, Weichao Mao, Saviz Mowlavi, Mouhacine Benosman, and Tamer Basar. ControlGym: Large-scale control environments for benchmarking reinforcement learning algorithms. *arXiv preprint arXiv:2311.18736*, 2023.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Yuming Zhou, Haoyang Li, Sheng Lin, Yanfeng Zhao, Tong Zhao, Xupeng Miao, Jie Jiang, Fangcheng Fu, and Bin Cui. Harnessing routing foresight for micro-step-level moe load balancing in rl post-training, 2026.

John A. ZuHone, Veronica Biffi, Eric J. Hallman, Scott W. Randall, Adam R. Foster, and Christian Schmid. Simulating X-ray observations with Python. In Stéfan van der Walt and James Bergstra (eds.), *Proceedings of the 13th Python in Science Conference*, pp. 98–104, 2014. doi: 10.25080/Majora-14bd3278-010. URL `https://proceedings.scipy.org/articles/Majora-14bd3278-010`.

A Extended scientific context and related work

The complete conceptual overview (Figure 1) and scientific context are retained here.

A.1 Scientific experience and acceleration

Language-model capability grows with the experience available for learning: text supports chat intelligence through human knowledge and instruction following (Ouyang et al., 2022; Christiano et al., 2017), while repositories, compilers, and tests make coding actions observable and outcomes checkable, letting models act, observe consequences, and learn from failures (Jimenez et al., 2023; Pan et al., 2025; Jain et al., 2025; Yang et al., 2025). Such experience also enables reinforcement learning with verifiable rewards (DeepSeek-AI et al., 2025; Wei et al., 2025). However, scientific intelligence remains challenging: whereas coding tasks usually specify a problem, goal, and acceptance test, discovery-oriented work asks agents to identify worthwhile questions, test hypotheses through interventions, and transfer lessons from evidence (Silver & Sutton, 2025; Sutton & Barto, 2018). Science concentrates these demands in sustained interactions with code, data, and simulators, where validated numerical models supply externally checked training signals.

Turning this opportunity into training experience exposes a scientific experience bottleneck. Heterogeneous toolchains and configurations make scientific repositories difficult to execute reproducibly; verification depends on physical quantities, numerical tolerances, and unwritten conventions; and runnable code must become meaningful tasks with behavioral validation, graded feedback, and recorded interactions before it can support learning. Thus papers, repositories, and datasets do not automatically yield trainable environments. Scientific coding and reproducibility benchmarks (Tian et al., 2024; Chen et al., 2025; Siegel et al., 2024; Xu et al., 2026) evaluate agents but do not supply the production infrastructure that expands experience across repositories and task families.

The bottleneck is sharpest where the payoff is largest. Moving scientific codes to accelerators still proceeds one hand-written port at a time (Rossazza et al., 2025; An et al., 2025), each bound to a vendor toolchain (Romeo et al., 2026) and to double precision as a default rather than a derived requirement (Düben & Palmer, 2014; Düben et al., 2015). ScienceIDE fixes the contract that already exists in a code's own tests and expert tolerances as executable checks per environment and decouples it from task authoring, so any task, from injecting a defect to porting an expensive path, is graded by whether it recovers the science within tolerance. The registry so far holds 2,515 repair, 295 implementation, and two acceleration tasks; on the hard subset, current systems exhaust the hour on up to a third of attempts and miss codebase-specific conventions in most failures (§4), the long-horizon work this infrastructure exists to train.

ScienceIDE is open infrastructure that makes scientific expertise reusable. Domain experts define executable repository environments, scientific cases, and calibrated checking policies, while AI agents help instantiate them; environment-specific factories reuse these decisions to generate valid tasks. Execution and scientific verification accept candidate tasks, and agent interactions over them yield graded trajectories; common interfaces provide supervised fine-tuning (SFT) data, online reinforcement-learning (RL) rewards, and evaluation with tasks or environments held out from training. Expert effort therefore concentrates on scientific boundaries and new objectives rather than each generated task. One environment thus serves repair, implementation, reproduction, and acceleration alike. An open registry lets users train and evaluate agents on a chosen domain and return them to scientific work, where their interactions supply new tasks (Figure 1b,c).

A.2 Extended related work

AI and science interact in two complementary directions (Figure 1b,c). *AI for Science* uses models and agents for scientific prediction, computation, and discovery; *Science for AI* draws on scientific ideas, data, and environments to develop AI capabilities. A scientific result does not necessarily improve the underlying model, just as a high benchmark score does not establish a reusable training environment. Connecting these directions requires more than successful task execution: scientific workflows must expose actions and outcomes that can be validated, reused, and converted into learning experience.

A.2.1 AI FOR SCIENCE: MODELS, AGENTS, AND EVALUATION

Specialist scientific models. Scientific models span protein structure prediction (Jumper et al., 2021), PDE learning through neural operators and physics-informed networks (Li et al., 2020; Raissi et al., 2017), weather forecasting (Lam et al., 2022; Bi et al., 2022; Pathak et al., 2022), materials generation (Zeni et al., 2023), symbolic regression (Udrescu & Tegmark, 2019), and scientific language and reasoning (Taylor et al., 2022; Lewkowycz et al., 2022). These approaches develop domain capabilities through specialized models, objectives, or corpora; agent systems additionally organize model actions into scientific workflows.

Scientific agents and discovery. Reported advances cover mathematics, physics, and biology. Examples include Claude's computer-checked Fermat formalization (Anthropic, 2026); GPT-5.2's gluon-amplitude conjecture, subsequently proved, and GPT-5's contributions to an Erdős problem and an experimentally confirmed immune-cell mechanism (OpenAI, 2026; Bubeck et al., 2025). AlphaEvolve extends FunSearch's program-search approach (Romera-Paredes et al., 2024), reporting an improvement over Strassen's 4×4 complex-valued matrix-multiplication algorithm and advances in Ramsey bounds (Novikov et al., 2025; Nagda et al., 2026). Biological examples include co-scientist drug-repurposing hypotheses (Gottweis et al., 2025), Robin's ripasudil proposal for macular degeneration tested in ARPE-19 retinal pigment epithelial cells (Ghareeb et al., 2025), and discoveries reported by Kosmos (Mitchener et al., 2025). Broader research frameworks pursue autonomy through tool use, search, orchestration, and collaboration (Lu et al., 2024; Yamada et al., 2025; Schmidgall et al., 2025; Kon et al., 2025; Bran et al., 2023; Boiko et al., 2023; Ghafarollahi & Buehler, 2024). These results demonstrate several forms of scientific validation, from formal checking to experimental tests. ScienceIDE focuses on making such execution and verification interfaces reusable across tasks and models.

Measuring scientific ability. Evaluation settings differ in task source and verification. Expert-authored or publication-derived suites include SciCode (80 problems) (Tian et al., 2024), ScienceAgentBench (102 tasks) (Chen et al., 2025), LAB-Bench (Laurent et al., 2024), the systems-biology SciGym (Duan et al., 2025), and DiscoveryBench (Majumder et al., 2024). Research-level evaluations span ML engineering (Chan et al., 2024), expert-referenced R&D (Wijk et al., 2024), frontier mathematics (Glazer et al., 2024), and broad expert questions (Phan et al., 2025); CORE-Bench and PaperBench instead grade the reproduction of existing results (Siegel et al., 2024; Starace et al., 2025). Closest to our software substrate, AInsteinBench sources maintainer pull requests from six scientific repositories, with expert review, annotated difficulty, and a requirement for well-structured test suites (Duston et al., 2025). SWE-bench Science covers 119 tasks from 98 repositories and emphasizes how software tests can under-specify scientific contracts (Xu et al., 2026). Expert authoring cost, saturation, and public-answer contamination motivate reusable task-generation and verification mechanisms alongside these evaluation resources.

A.2.2 SCIENCE FOR AI: IDEAS, DATA, AND LEARNING ENVIRONMENTS

Scientific foundations and supervision. Science contributes theoretical ideas, training data, and evaluation problems. Statistical mechanics informs energy-based models, while nonequilibrium thermodynamics inspired diffusion models (Sohl-Dickstein et al., 2015). Structural, atmospheric, and materials databases support the specialist models above, and scientific corpora support language-model pretraining. Graduate-level science and research-level mathematics provide reasoning evaluations such as GPQA, FrontierMath, and HLE (Rein et al., 2023; Glazer et al., 2024; Phan et al., 2025). Evaluation on these problems does not itself establish that they supplied training gradients. Interactive learning adds another channel: an agent acts, observes consequences, and receives feedback from an executable environment.

Executable environments and the coding precedent. The learning stack separates environments, action interfaces, and optimization. Game, procedural, embodied, web, and multi-agent settings provide interaction (Bellemare et al., 2012; Brockman et al., 2016; Towers et al., 2024; Cobbe et al., 2019; Küttler et al., 2020; Hafner, 2021; Fan et al., 2022; Côté et al., 2018; Shridhar et al., 2020; Zhou et al., 2023; Terry et al., 2020; Lanctot et al., 2019); algorithm libraries support training (Raffin et al., 2021; Huang et al., 2022; Bou et al., 2023). Language-model agent suites (Stojanovski et al., 2025; Guertler et al., 2025; Abdulhai et al., 2023; Yao et al., 2024; Liu et al., 2023; Mialon et al.,

2023) complement training and inference systems (Hu et al., 2024; Wang et al., 2025; Liu et al., 2024; Tan et al., 2025; Cao et al., 2025; Sheng et al., 2024; Kwon et al., 2023; Rasley et al., 2020) and verifier-based supervision (Lightman et al., 2023; Zelikman et al., 2022; Singh et al., 2023). Tool-use scaffolds enable reading, editing, and execution (Yao et al., 2022; Shinn et al., 2023; Wang et al., 2024a; Yang et al., 2024; Wang et al., 2024b; 2023). Within coding, SWE-bench established repository evaluation (Jimenez et al., 2023); SWE-Gym provides 2,438 real tasks with runtimes and tests (Pan et al., 2025), R2E-Gym supplies 8.7k procedural tasks with hybrid verifiers (Jain et al., 2025), debug-gym exposes interactive debugging (Yuan et al., 2025), and SWE-smith synthesizes 50k task instances (Yang et al., 2025). SWE-RL learns from software evolution (Wei et al., 2025), alongside broader reasoning and agent training with verifiable feedback (DeepSeek-AI et al., 2025; Team et al., 2025; Yu et al., 2025; OpenAI et al., 2024; OpenAI, 2025; Anthropic, 2025). Formal mathematics provides a related example through proof-checker feedback (Xin et al., 2024). Together these directions give concrete mechanisms for learning from experience (Silver & Sutton, 2025).

Scientific interaction environments. Existing scientific learning settings expose different kinds of action: RL for tokamak control (Degrave et al., 2022), molecular generation with task-specific rewards (Bengio et al., 2021), and physics-simulator feedback for Olympiad reasoning (Prabhudesai et al., 2026). Solver-control gyms expose dynamical systems for control (Lagemann et al., 2026; Bhan et al., 2024; Zhang et al., 2023), with safe-control environments studying constraints on action (Ray et al., 2019); GEM, MLGym, Aviary, and DiscoveryWorld provide broader agent interaction settings (Liu et al., 2026; Nathani et al., 2025; Narayanan et al., 2024; Jansen et al., 2024). Scaling such experience across scientific codebases requires accounting for simulation cost, numerical variation, and multiple valid outcomes. A verifier must specify which scientific quantities to compare, over which workloads and tolerances, and how partial progress becomes reward. These requirements extend the environment-scaling agenda developed for code, terminals, browsers, and games (Huang et al., 2025; Chen et al., 2026).

Positioning ScienceIDE. Our comparison centers on task source, scientific verification, reuse, and the learning interface. Benchmark instances provide a unit of evaluation; ScienceIDE instead makes an expert-approved scientific module and its checks a reusable unit for producing task instances. Environment-specific factories adapt authoring procedures to that unit, and evaluation, SFT, and RL consume the resulting interactions. The contribution is this connection between scientific curation, repeated task production, and model learning, with scientific content retained across changes in task family or trainer.

B Extended environment and task construction

B.1 System objects

ScienceIDE turns scientific expertise into reusable agent experience. Experts define scientific responsibilities and acceptance criteria; executable environments preserve these decisions, task factories generate challenges, and agent interactions provide evidence for evaluation and learning (Figure 2).

The construction unit is a scientific *module* within a versioned *codebase*. A module owns a coherent scientific responsibility and executable coverage, rather than simply grouping related files. An *environment* packages an approved module with its runtime and scientific checks. Scaling adds modules and codebases while reusing their acceptance criteria across task families.

A *factory* specializes authoring procedures to an environment. Its validated *tasks* specify an initial workspace, a deliverable, and a verifier. An *episode* is one agent interaction with a task, producing artifacts, a trajectory, and measured outcomes. The following stages explain how these objects are constructed and connected.

B.2 Compiling scientific expertise into executable environments

Construction begins with production scientific software (Figure 7). An agent inspects a pinned upstream revision, dependencies, licence, and build assumptions, then builds the source and runs

official tests and examples. These runs expose output formats, numerical variability, expensive paths, and execution hazards that inform the environment boundary.

The agent proposes modules defined by scientific responsibility and executable coverage. Each module identifies its inputs and outputs, algorithm stages, owned implementation paths, excluded responsibilities, and supporting tests. Shared solvers and toolchains may support several modules without erasing their scientific distinctions. A domain expert reviews the decomposition and coverage, allowing one repository to contribute several environments with traceable source identity.

The agent implements the environment-specific experiment adapter and proposes scientific outputs for review; the curator owns module boundaries and the equivalence contract. The registry and CLI enforce structure and track provenance without choosing scientific observables. An approved module is packaged with an editable workspace, checks, and a private verifier. Source pins, dependencies, and observed hazards remain attached to this reusable runtime.

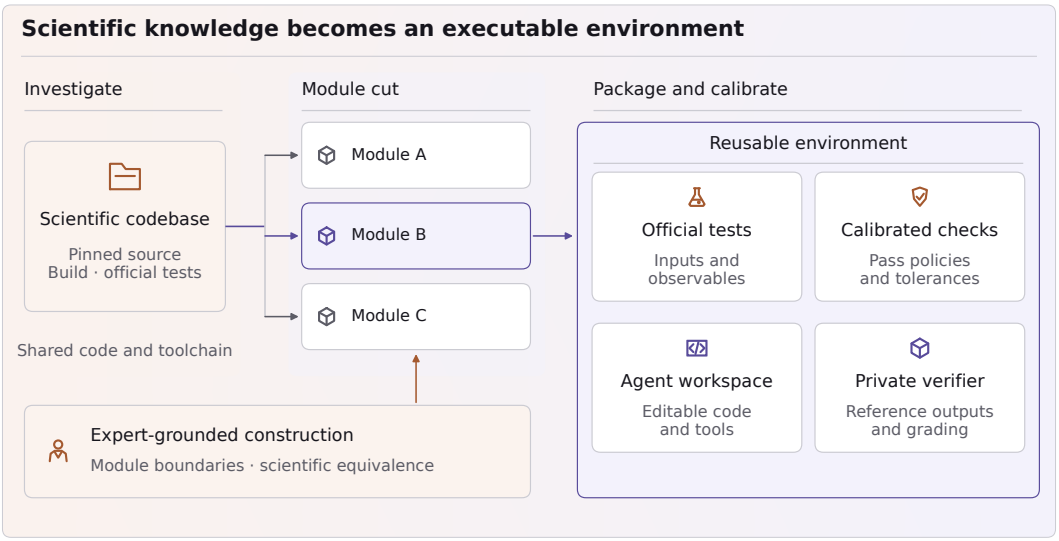


Figure 7: **From a codebase to an executable scientific contract.** Experts define module boundaries and scientific equivalence. Each selected module packages official tests with calibrated pass policies, an editable agent workspace, and a separate private verifier. These assets are reused across task families rather than reconstructed for each task.

B.3 From official tests to scientific checks

Once a module is approved, its upstream unit tests, regression tests, and shipped example problems are surveyed together. Each case is traced to the module responsibility, run or marked as unmeasured, and either retained as a check, explicitly excluded with a reason, or identified as a gap. The unit of reward is a *check*: fixed inputs and graded outputs paired with a *pass policy*. An example remains an official test even without a shipped reference: the pinned build supplies the comparison and the example's physics (a published value, convergence, or a conserved quantity) anchors it. A check with no official source is *custom* and requires curator agreement.

The pass policy has two forms. A *pointwise* policy compares every graded value, $|c - r| \leq a + \rho\,|r|$; exact equality is the special case $a = \rho = 0$. Pointwise is preferred whenever a bound contains the measured sensitivity over a scientifically meaningful window while still rejecting a real fault. It grades physical observables, never bookkeeping: an unordered collection is aligned by an identity carried in the output before comparison (a particle's properties follow its particle ID rather than its array slot), and the same alignment covers its associated arrays. Storage order, adaptive step counts, timings, rank or chunk layout, random draws, and eigenvector phase are not observables. An *invariants* policy compares quantities such as moments, distributions, conserved values, or integral norms when no pointwise bound can contain the run-to-run variation and reject a fault. This includes random streams, rapidly diverging flows, sampling statistics, and discrete

outputs. The graded window is shortened first when the physics still survives that choice; otherwise invariants are used from the start.

Nominal and variant initial conditions make this choice measurable. The variant perturbs the smallest sufficient set of active inputs so that the graded outputs reveal numerical sensitivity; it is calibration evidence, not a physics-isolation experiment or an automatic tolerance rule. Where a build permits it, an optional *altbuild* runs the nominal input on another legitimate build of the same pinned source and records the observed cross-build separation. A check without such a build has no altbuild floor; even a measured floor or nominal–variant spread is finite evidence, not a universal guarantee. Self-validation runs nominal and variant independently, requires each to reproduce the reference and attain full reward, and records their observations separately. The curator then finalizes the policy, tolerance, window, and variant by reading the source and its numerical mechanisms. Each check carries a plain-language *warrant*: it identifies the observable, explains which scientifically relevant bias the bound distinguishes, and explains why a valid implementation can satisfy it. Implementation-level calibration details are given in Appendix C. The curator and domain expert then review the package over fresh rounds, reproducing and revising checks when needed; revised contracts require fresh evidence before acceptance.

The check suite is fixed before downstream task authoring. Task statements can therefore vary the requested work while every check remains part of the acceptance contract.

B.4 Specializing reusable methods into task factories

A factory combines reusable authoring procedures with an environment's scientific context (Figure 8). Shared procedures handle reversible edits, execution, and artifact assembly; local rules identify active paths, cases, build recipes, and meaningful transformations. The packaged module map, observables, tolerance evidence, and known limitations constrain candidate generation.

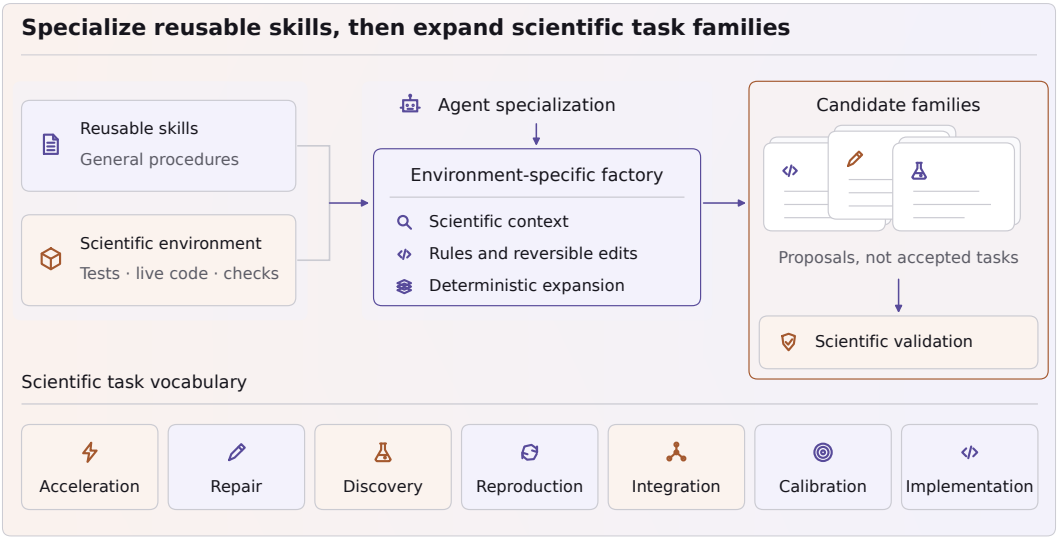


Figure 8: **General skills become environment-specific task factories.** An agent adapts reusable procedures to a scientific environment; local rules expand reproducible candidates, which become tasks only after later scientific validation. The seven categories describe the authoring taxonomy, not the measured coverage of deployed generators.

The agent proposes semantic edit sites and objectives, while the curator and domain expert retain decisions about observables, equivalence, and acceptance. Deterministic procedures expand approved rules into mutation or excision candidates; the validation stage determines which become tasks. For example, a LAPS environment can reuse its scientific checks for both defect repair and reconstruction of a missing timestep routine. The requested work changes, while the numerical behavior to be recovered remains defined by the same environment.

The authoring taxonomy has seven categories: Acceleration, Repair, Discovery, Reproduction, Integration, Calibration, and Implementation. Repair and Implementation expose automatable candidate expansion through reversible mutation and excision. The other categories provide interfaces for expert-specified objectives, data, procedures, or resource constraints. Verification combines reference equivalence, conformance tests, bounds on scientific quantities, and re-execution as appropriate to the task. Acceleration adds a resource criterion after correctness; reproduction grades the outputs of a runnable procedure. This division reuses mechanical authoring operations without transferring scientific judgment to them.

B.5 Manufacturing and validating scientific tasks

Factory proposals become tasks only after executable evidence establishes that they are observable, solvable, and trustworthy (Figure 3). AI authors can propose repairs, implementations, accelerations, reproductions, or other transformations, but a proposal is not evidence of validity.

Propose broadly; establish validity by execution. Factory rules turn AI-proposed transformations into candidates, including syntactic mutations (Just et al., 2014b;a), cross-component changes, and non-repair objectives. Each candidate is tested in the final grading environment: its objective must be attainable and yield an informative score. For an injected repair, a known-valid witness must pass, the unfixed baseline must leave headroom, and the change must produce a check failure removed by the reference repair. Where applicable, compilation, native execution, and coverage precede admission; family-specific closed-book and visibility probes provide design evidence rather than universal validity criteria. Silent mutations may remain explicit controls, whereas ambiguous specifications, unreachable branches, and harness failures are invalid or ungraded.

Trustworthy packaging and useful progress. Scientific checks expose partial accomplishment; repair scores normalize it against the defective starting point using Eq. (1), where r measures scientific agreement and f is the unfixed build's score. Packaging also screens answer leakage and records execution-integrity failures; the gate and self-test protocol is part of task construction. For objectives other than repair, evidence must show an attainable target, observable outcome, and informative credit for alternatives. The task record separates graded outcomes from execution and infrastructure failures and preserves the failure reason for audit.

Policy- and budget-relative difficulty. Design aims for clear specifications, substantial reasoning depth, legitimate alternatives, and resistance to leakage; calibration and probe considerations are summarized in Appendix C.

After validity, task-family admission labels remain distinct from difficulty measured for a named model or panel, scaffold, budget, and date. Trace audits distinguish solver failure and budget exhaustion from infrastructure faults, ambiguous specifications, and reward misalignment; each model diagnosis needs supporting evidence, and no judge can override deterministic anchors. Outcomes are reported in Experiment and feed back into factory rules, cases, and budgets. Design labels remain design intent, not retrospective claims about measured hardness.

B.6 Connecting scientific interaction to model learning

A validated task exposes a common episode interface (Figure 9). The agent receives an editable workspace, inspects code and scientific inputs, makes changes, runs experiments, and submits artifacts to a private verifier. The harness records actions, observations, check rewards, execution status, and resource use, distinguishing scientific disagreement, incomplete delivery, and infrastructure failure.

Evaluation, offline supervision, and online learning. Evaluation holds scientific tasks and interaction budgets fixed while measuring success, partial progress, and resource use. SFT consumers select trajectories and supervise agent actions, including tool calls; RL consumers connect policy-controlled rollouts to verifier rewards. Tasks can be selected by environment, domain, family, or measured difficulty for held-out evaluation and training curricula. Models and trainers can change without rebuilding the scientific content or its acceptance criteria.

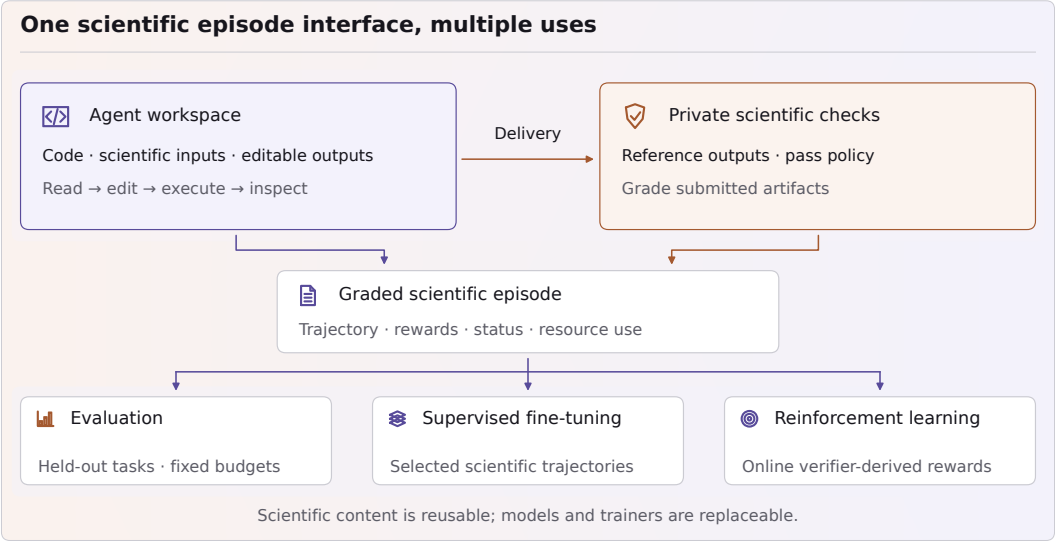


Figure 9: **A shared scientific episode interface supports evaluation and learning.** Agent workspaces submit artifacts to private scientific checks. The episode records trajectories, rewards, execution status, and resource use. Evaluation, SFT, and RL consume this experience through separate interfaces; scientific content remains reusable across models and trainers.

Returning capability to science. The environments also support maintenance, acceleration, reproduction, tool integration, and discovery-oriented work. Interactions supply trajectories and proposals for new objectives; proposed tasks or checks re-enter scientific validation before joining the registry. The same interface thus supports learning from scientific work and applying the resulting agents to further work.

C Scientific contracts and experience inventory

C.1 Scientific check calibration

A check fixes its inputs, graded outputs, and a pass policy. Pointwise is preferred when measured sensitivity fits a meaningful window and the bound distinguishes a scientifically relevant deviation. Outputs whose storage order may change are aligned by an identity carried by the output; timings, adaptive-step counts, rank layout, and random draws are not observables. Invariants are used when pointwise comparison cannot contain valid variation without losing discrimination.

Nominal and variant initial conditions provide calibration evidence: each run is executed independently, must reproduce its reference and attain full reward, and contributes a measured spread. Where the build permits it, an optional alternative legitimate build of the same pinned source records a cross-build separation. These observations are finite evidence, not a universal floor or an automatic tolerance. The curator and domain expert choose the policy, bound, window, and variant by reading the source mechanism and the scientific meaning of the observable. Each check carries a warrant that states what is compared, which scientifically relevant deviations the bound is intended to distinguish, and why valid implementations can satisfy it. Fresh review can revise the check before acceptance.

Once fixed, the check suite is shared by repair, implementation, acceleration, and reproduction families. A factory may vary the agent's initial state, deliverable, and task-specific objective, but it does not silently redefine scientific equivalence. Thus the expert work invested in a module's checks is amortized across task families while acceptance remains executable.

C.2 COHORT-BOUND EXPERIENCE INVENTORY

The inventory keeps denominators attached to the population that produced them. The agent-evaluation cohort, scientific-contract cohort, and task-supply cohort support different claims and must not be pooled: task supply and check coverage do not change the ScienceIDE-Hard evaluation denominator.

Table 1: **Cohorts kept separate in the experience inventory.**

Cohort	Scope	Interpretation
Agent evaluation	85 tasks across 18 environments	Fixed denominator for ScienceIDE-Hard model comparison.
Scientific contracts	1,076 executable checks	Broader observable-equivalence and invariant coverage.
Task supply	2,515 repair, 295 implementation, 2 acceleration	Manufactured task inventory, not an evaluation denominator.

The table is an accounting boundary, not a claim that every environment has the same task families or check density. Source citations and license records remain in Appendix J; repository records are designated for anonymous supplementary material.

Probing accounting. The policy-era escalation used one reference probe first and retried failures up to three attempts: 2,722 attempts versus 4,845 under a uniform three-probe policy, or 43.8% fewer. Across the pooled campaign, 3,195 graded trajectories cost an estimated $703 (about $0.22 per trajectory), a subscription-metered probe cost rather than an end-to-end construction cost. These comparisons are descriptive, not preregistered. The rate of mixed-outcome tasks missed by single-success screening remains unaudited.

D EVALUATION PROTOCOL AND INTEGRITY

Fixed scientific cohort. ScienceIDE-Hard contains 85 tasks from 18 environments and five repository families: 52 repair and 33 implementation tasks (MITgcm 33, PLUTO 31, LAPS 11, Athena++ 9, PHANTOM 1). Every leaderboard model receives the same task identifiers. Each admitted task has a known-valid witness, empty-delivery baseline, and measured defective starting point; strict success requires the private scientific checks, not merely compilation or execution.

Harnesses, budget, and repeats. Agents use Codex, Claude Code, or Gemini CLI with identical task containers and instructions, no localization hints, and a one-hour budget; results remain model–harness measurements. The analysis contains 3,982 validity-filtered attempts. Fourteen models have three valid attempts on every task, while Fable uses its latest valid attempt. Scores average within task and then weight tasks equally. Intervals are repeat-bootstrap intervals; budget curves use 1,200 seeded replicates and are retrospective, not shorter-budget reruns. Fable has no interval.

For model m, the budget profile retained in the full analysis is

$$\widehat{p}_m(b) = \frac{1}{N}\sum_{t=1}^{N}\frac{1}{K_{mt}}\sum_{i=1}^{K_{mt}}\mathbf{1}[S_{mti}=1,\, T_{mti}\le b], \quad N=85, \tag{3}$$

where T is recorded duration capped at one hour; this is not a shorter-budget rerun.

D.1 VALIDITY AND EXECUTION-INTEGRITY AUDITS

The reference-probe cohort contains 1,858 tasks across 25 environments and five repository families; it files tiers but does not define the 85-task comparison denominator.

Anchors, floors, and infrastructure errors. The reported core is validity-filtered, with oracle/witness, empty-delivery, and defective-floor anchors checked. The earlier bank audit was post

hoc: verifier crashes, missing references, OOMs, malformed rewards, and grader errors are infrastructure errors excluded from the denominator, whereas a valid `AgentTimeoutError` is budget exhaustion and counts as failure. The historical census found 1,705 effective tasks, 136 without headroom, and 322 with unverified floors; this conditional audit is not certification of the current full registry and does not replace the 85-task denominator.

Container-side retrieval. The original campaign declared network isolation and an action-anchored fetch audit, but later trajectory inspection found 455 upstream-directed commands among 5,314 trials, with 21 confirmed successful fetches and 13 scored as solved. Nineteen staging scripts had rewritten the agent-side declaration to `public`; the repository task files therefore did not describe the containers that actually ran. These are historical contamination findings, not evidence that the earlier campaign was controlled. The corrected path uses a live-container allowlist and voids confirmed fetch trials.

D.2 Provider and credential channels

Container isolation does not cover provider-side search/fetch or an aggregator credential that can call another browsing-capable model. Gemini received substantive provider-tool content on 71 of 85 tasks; one trial retrieved an exact 22,180-character upstream file through `web_fetch`, and affected trials were rerun with provider tools disabled. DeepSeek V4 Pro used an aggregator key to call other models on 20 tasks, solving 13; a provider-side model allowlist was required before rerunning. In that historical correction, the sample score moved from .365 to .294; this is not a timeless current leaderboard value. These fixes close the applicable channel but do not claim pretrained models never saw public upstream code.

D.3 Effort accounting and numerical details

Cost analyses use recorded harness estimates for nine models and archived input/cached-input/output rates for six; cached input is subtracted first. Two missing token totals came only from final structured usage records, and missing native costs are not inferred. Detailed leaderboard, budget, efficiency, census, and answer-path tables remain frozen source artifacts. Historical difficulty exploration establishes only that zero can mean a broken task, budget exhaustion, retrieval, reward misalignment, or solver failure; these are not standalone causal labels.

E Extended budget and resource results

Agents differ in how quickly they solve scientific tasks. At ten minutes, Astra reaches 49.6% success against Fable's 25.9%; Fable overtakes it at approximately 31 minutes (Figure 4). Between 20 and 60 minutes, Astra gains only 1.9 percentage points, compared with 11.8 for Fable, 16.0 for Opus, and 30.2 for Qwen3.8 Max. Similar final scores can therefore conceal substantially different time requirements.

More expenditure does not guarantee greater success. On the leaderboard cohort, Fable achieves 67.1% success at an estimated $7.90 per task, while Astra achieves 63.1% at $3.56 (Figure 10). Astra averages 9.4 minutes and 13.9k output tokens per task, compared with Fable's 16.8 minutes and 85.7k tokens. DeepSeek V4.1 Flash produces 172.4k output tokens per task for 36.0% success. Across the fifteen profiles, the descriptive Spearman correlations of success with runtime and output volume are -0.22 and 0.01. Resource use and scientific correctness are thus distinct dimensions of performance.

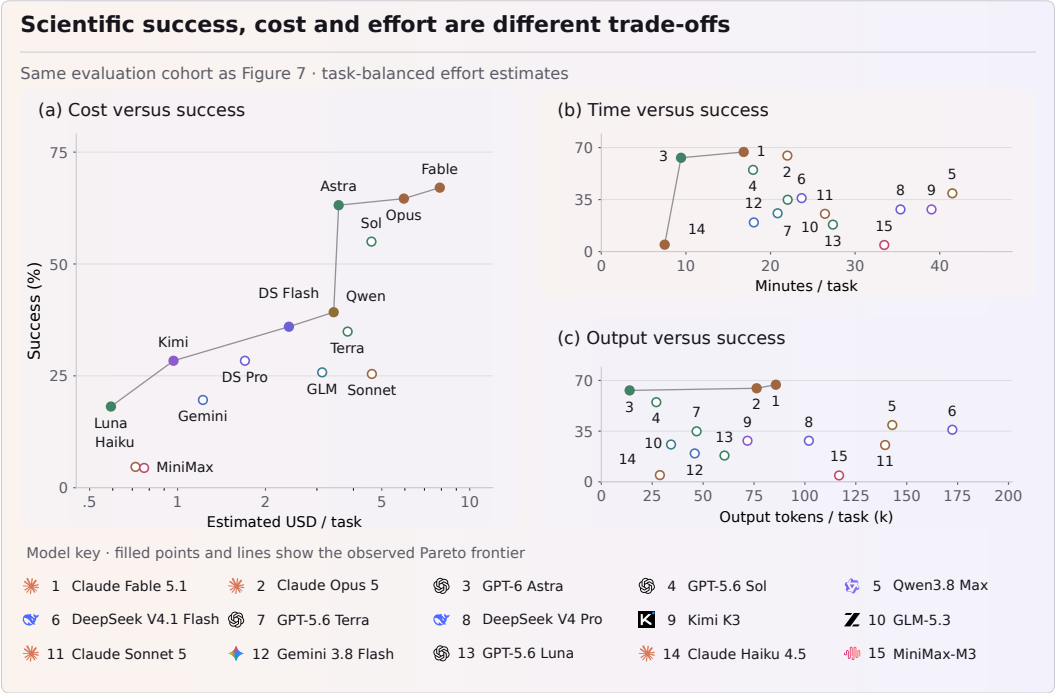


Figure 10: **Outcome–cost profiles separate accuracy from expenditure.** Success rates match the leaderboard for all fifteen agents. Effort averages available telemetry within each task from the same selected measurements. Filled markers and connecting lines show observed Pareto frontiers. Costs use recorded harness estimates or archived token rates rather than current prices.

F AGENT ERROR ANALYSIS AND SCIENTIFIC COVERAGE

This appendix complements the aggregate success, budget, and cost results with failure behavior, scientific specialization, and trajectory-level evidence on ScienceIDE-Hard.

F.1 FAILURE OUTCOMES AND REPEAT INSTABILITY

Low success can reflect very different execution behavior. Qwen exhausts the budget on 37.3% of its selected attempts and spends 48.5 minutes on an average unsuccessful episode. Haiku has no recorded budget timeouts, averages 7.7 minutes on unsuccessful episodes, and returns at or below baseline on 74.9% of attempts. MiniMax combines low success with 31.6% budget exhaustion and 34.1-minute unsuccessful episodes (Figure 11). Failure can thus involve either brief unsuccessful work or prolonged effort without a solution; termination status alone cannot establish why an agent stopped.

Success is also unstable across repeated execution. Among agents with complete coverage, 28.0% of model–task pairs contain both success and failure within their first three valid attempts. Gemini changes outcome on 33 of 85 tasks, compared with eight for Astra and 29 for Kimi. An aggregate success rate can therefore conceal unreliable task-level behavior. The trajectory analysis below examines what differs between successful and unsuccessful attempts.



Figure 11: **Failure behavior and repeat instability are distinct diagnostics.** Task-balanced shares of success, budget exhaustion, non-timeout partial credit, and baseline-level returns, with mean unsuccessful-episode duration and first-three-repeat outcome instability. Error bars require complete repeat coverage. These are observable outcomes, not inferred causes of reasoning failure.

F.2 Scientific specialization and complementarity

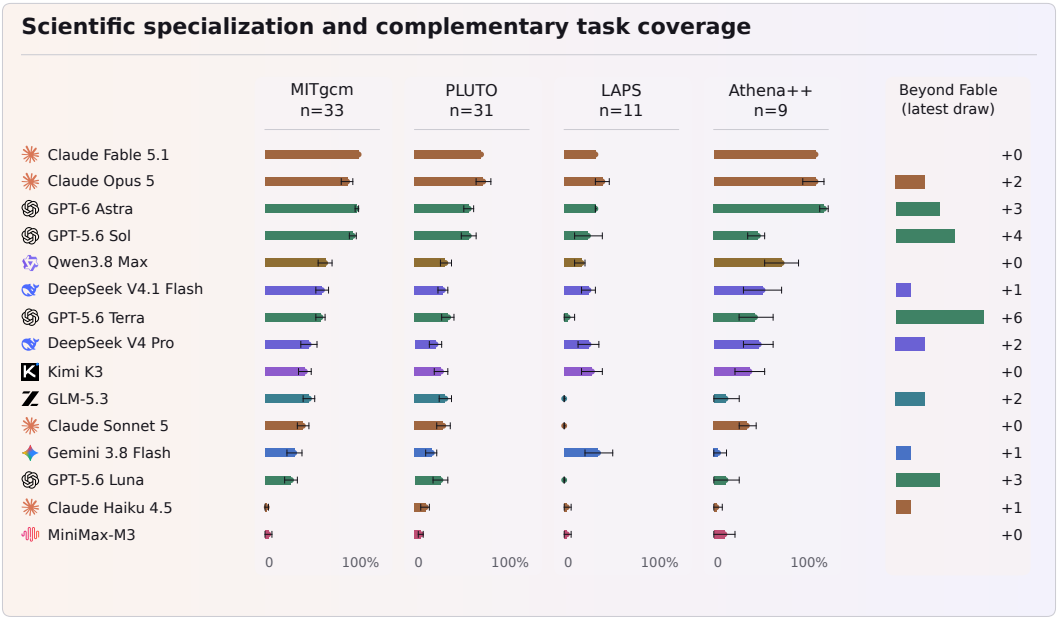


Figure 12: **Aggregate rank does not summarize scientific coverage.** Task-balanced success by repository family and additional successes beyond Fable in a common single-attempt sample. Error bars summarize repeated execution where coverage is complete. PHANTOM's single task is excluded from the family panels but retained in the 85-task coverage analysis. Additional solved sets may overlap.

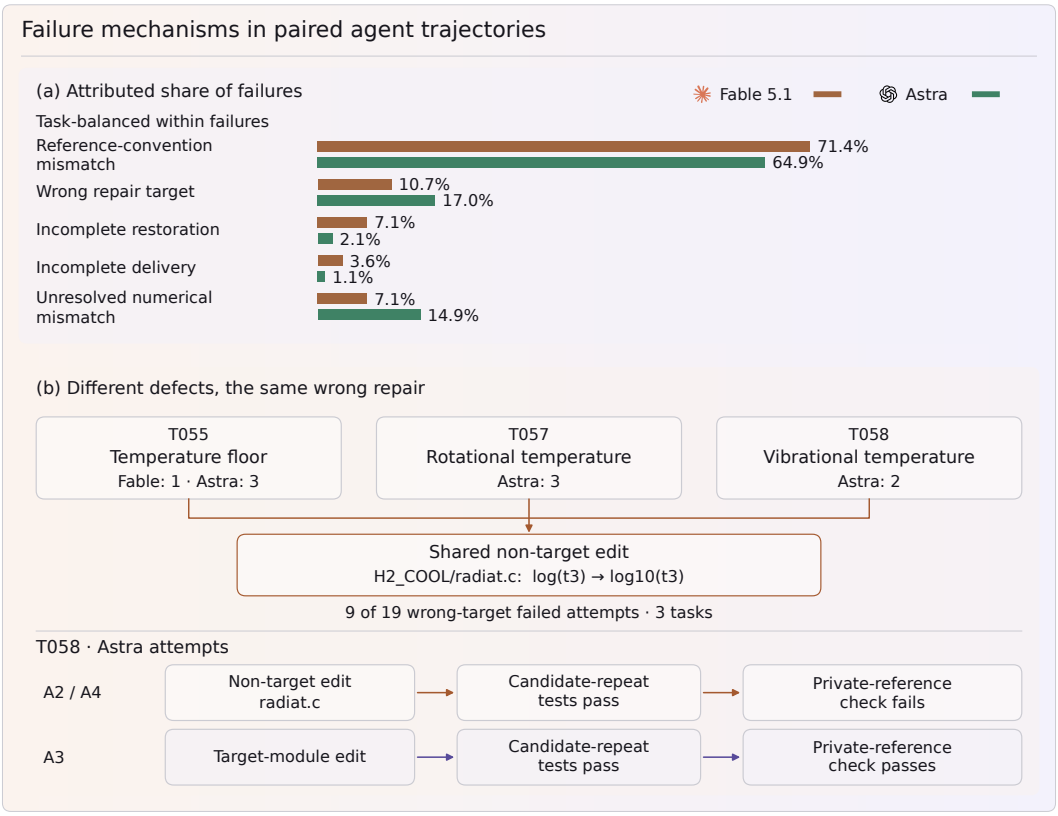


Figure 13: **Different defects can lead to the same non-target repair.** (a) Primary mechanism annotations, task-balanced within unsuccessful attempts. (b) Nine failed attempts across three tasks converge on one non-target edit; a same-task Astra success repairs the intended target. Candidate-repeat checks pass on both paths, but private-reference outcomes differ. A2–A4 identify individual attempts.

Scientific capabilities are uneven and complementary. Astra and Sol have nearly equal PLUTO success rates (47.3%), but differ substantially on Athena++ (96.3% versus 38.5%). Gemini's LAPS rate (29.5%) exceeds Sol's (20.8%), despite their opposite aggregate ordering (Figure 12). These reversals reveal uneven performance across scientific software; the families also differ in language and task composition, so the comparison does not isolate a domain effect.

In the common single-attempt coverage sample, Fable solves 57 of 85 tasks, while the union across all fifteen agents solves 72. Terra contributes six successes outside Fable's solved set despite its lower aggregate score. Eleven tasks are solved by exactly one agent, and 13 by none. The union indicates observed complementarity, not the performance of a tested ensemble: realizing it would require a selection or verification policy and additional computation.

F.3 Trajectory-level failure mechanisms

We examine why agents fail by reviewing Fable 5.1 and Astra on all 85 ScienceIDE-Hard tasks, including successful repeats as within-task contrasts. The review covers 358 attempts, of which 125 are unsuccessful. A model-assisted reviewer inspects original actions, tool outputs, and grading records for each task. The resulting annotations describe evidence-supported mechanisms, not experimentally isolated causes; the protocol and complete cases are in Appendix F.4.

Scientific reconstruction requires codebase-specific conventions. Reference-convention mismatches account for 71.4% of Fable's and 64.9% of Astra's task-balanced failures (Figure 13a). In a LAPS timestep task (T013), both models produce runnable code but omit the rule for retaining

the previous timestep. The lower-resolution case matches the reference, while the higher-resolution case diverges at a periodic update. In an MITgcm tracer task (T035), all four attempts reconstruct nearly the same coefficient table with one truncated coefficient; the associated tracer diverges while another matches. The challenge is therefore not only recovering a scientific formula, but reproducing the numerical precision and state conventions required by the surrounding code.

Different defects can induce the same wrong-target repair. Wrong-target edits account for 10.7% and 17.0% of the two models' task-balanced failures. Nine failed attempts across three PLUTO cooling tasks make the same natural-log to base-10-log change in a non-target routine, although the assigned defects concern three different quantities: a temperature floor (T055), rotational temperature (T057), and vibrational temperature (T058). The intended defects remain unrepaired. On T058, a successful Astra repeat instead restores the target constant in the equation-of-state module (Figure 13b). This contrast identifies a recurring localization error rather than an isolated implementation typo; it does not establish why the non-target expression attracts the models.

A passing local test may not test the assigned task. Both the unsuccessful and successful T058 repairs pass comparisons between repeated runs of their own candidate code. Only the target repair passes the private-reference evaluation. Similarly, failed Astra attempts on a PLUTO substep-handoff task (T059) test a modified error-clamp routine while leaving the defective handoff unchanged; the successful repeat repairs the handoff. The local tests establish repeatability or behavior of the chosen edit, not its relevance to the task. All three Astra attempts on T057 repeat the same non-target logarithm edit, and all three on T013 omit the same timestep-retention rule. Repeated execution can thus reproduce a specific mistake, not just independent failures with different explanations.

Correct code does not guarantee a complete scientific delivery. On an Athena++ chemistry task (T002), Fable and two Astra attempts apply the correct target edit, but omit four required control-output groups. Four other Astra attempts deliver the complete run set and pass. The failure is therefore incomplete scientific execution rather than an incorrect repair. Together with the localization contrasts, this case separates three requirements: selecting the relevant change, reconstructing the required behavior, and completing the deliverables. All eight mixed-outcome Astra tasks were included in the review; Appendix F.4 presents representative cases.

Seventeen unsuccessful attempts remain unresolved after targeted re-review, often because plausible numerical reconstructions still disagree with the reference. The annotations are neither expert-certified causal labels nor evidence that the verifier is wrong. For learning from these trajectories, the useful distinction is between the edit, what its local tests establish, the delivered artifacts, and the task-level outcome. Localization and delivery failures suggest different feedback; unresolved cases should retain their uncertainty.

F.4 Trajectory review protocol

We reviewed Fable 5.1 and Astra on the common 85-task panel, covering 358 attempts and 125 unsuccessful attempts, including successful repeats as within-task contrasts. A model-assisted reviewer read the original actions, tool outputs, and grading records. The annotations are evidence-supported mechanism labels, not experimentally isolated causes: the review was not blinded, independently replicated, or domain-expert certified, and 17 attempts remained unresolved. The task source files were not independently proven byte-identical to every historical revision, so code-level attributions retain that provenance limitation. No replayed command was executed.

The review is task-balanced rather than attempt-weighted. Repeated local tests show that an agent's chosen edit is reproducible, but they do not establish that it addresses the assigned defect; private-reference grading remains the acceptance test. Detailed task-level records and evidence references remain in the accompanying source artifacts.

G Extended learning experiments

G.1 Public-benchmark SFT results

We test whether scientific interaction trajectories support transfer to public benchmarks. We fine-tune Qwen3.5-4B, Qwen3.5-9B, and Qwen2.5-72B-Instruct on verified ScienceIDE demonstrations that record code inspection, tool use, and responses to execution feedback. Each SFT checkpoint is evaluated against its initial model under matched settings.

Demonstrations and supervision. We collect demonstrations with GPT-5.6-sol and select trajectories using the numerical-equivalence verifier. The training partition contains 4,567 segments from 564 tasks, and the validation partition contains 544 segments from 81 tasks. We retain the source partition assignments; no task identifier appears in both partitions.

Training and validation. We train with ms-swift, applying LoRA to all linear layers for three epochs, and evaluate the final checkpoints without selecting them on public-benchmark performance. Example construction, hyperparameters, and teacher-forced validation loss and token accuracy are reported in Appendix H.1. These diagnostics measure prediction of the demonstrations; scientific task execution requires a separate evaluation.

Public-benchmark evaluation. We evaluate the initial and SFT models on public benchmarks covering scientific knowledge, mathematical reasoning, code understanding, repair, and generation. Within each model pair, both checkpoints are evaluated on identical items. Scoring rules and generation settings are given in Appendix H.1; scores from different metrics are reported separately, without an overall average.

Gains span code, reasoning, and knowledge tasks. Figure 5(a) shows 15 model–benchmark comparisons with improved scores. The code gains span repair, execution prediction, defect detection, and generation. For 4B, HumanEvalFix JavaScript improves by 10.98 percentage points, and QuixBugs Java increases from 0.400 to 0.500 (+10.0 points). CodeXGLUE defect detection increases from 0.422 to 0.492 (+7.03 points), although final accuracy remains below 0.5.

Smaller gains of 3–6 points appear across code repair, execution prediction, and generation for 9B and 72B, and the improvements extend to reasoning and knowledge benchmarks: for 9B, BBH Word Sorting increases from 0.240 to 0.576 (+33.6 points), alongside gains on GSM8K, BBH multistep arithmetic, MMLU-Pro Computer Science, and ARC-Easy. Appendix H.1 walks through every comparison.

Across the reported comparisons, SFT on scientific interaction trajectories improves code, reasoning, and knowledge benchmark scores. These results support ScienceIDE as a source of verified demonstrations for model training.

G.2 RL execution and training dynamics

Scientific environments also provide a reward signal that supervised data cannot: a verifier scores an agent's repair by compiling the modified code and running the underlying scientific simulation. We use this signal directly as the RL reward and ask whether online optimization can improve Qwen3.5-4B beyond its base checkpoint. We train on two environments, LAPS, a 3D pseudo-spectral Hall-MHD solver (Shi et al., 2024), and MITgcm-biogeo, which models ocean biogeochemistry and CFC transport. Their task banks contain 99 and 87 repair tasks over three and nine graded scientific cases, respectively. Each repair is graded in $[0, 1]$ with partial credit. The instructions identify the defect's file, line, and edit class, leaving the agent to determine what the repaired code should compute.

This setting makes RL substantially harder than conventional tasks. A verifier verdict arrives only after the episode ends, so reward is outcome-only over trajectories that may span tens of turns and tens of thousands of tokens. Rollout and training run concurrently on disjoint GPUs, with token-level truncated importance sampling correcting the resulting one-step policy staleness (Appendix H.2).

Execution and training stack. Generation runs in vLLM (Kwon et al., 2023) and optimization in PSRL (Li et al., 2026; Zhou et al., 2026), a customization of the veRL trainer (Sheng et al., 2024) that

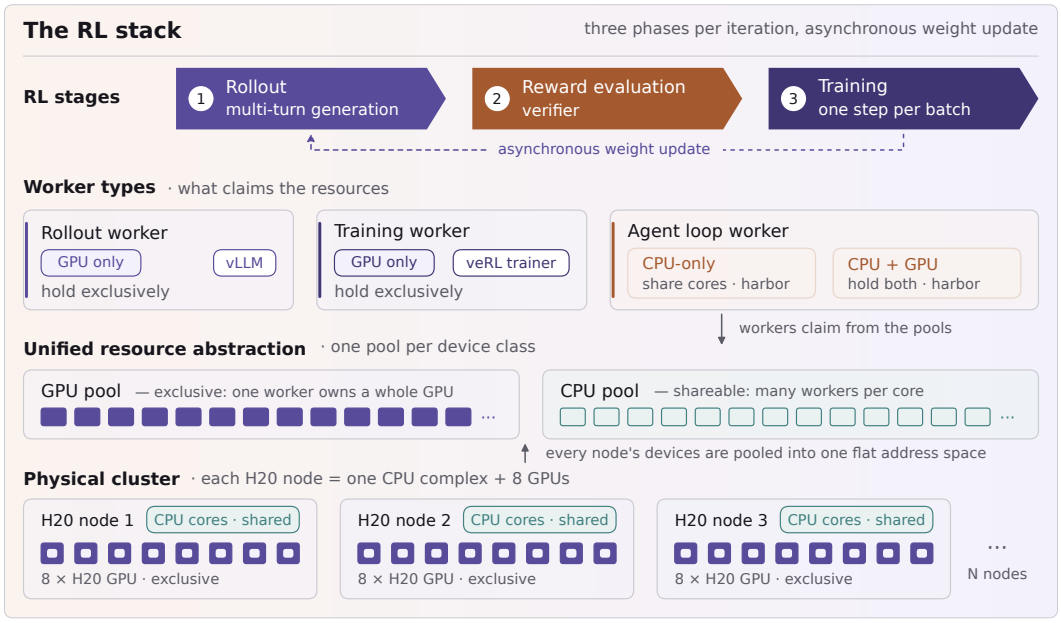


Figure 14: **The RL stack.** Three phases per iteration with asynchronous weight updates. Rollout workers (vLLM) and training workers (customized veRL trainer) exclusively occupy GPUs, while agent-loop workers (harbor) that drive episode execution either share CPU cores or exclusively occupy both CPUs and GPUs.

drives a black-box harness; episodes execute in harbor through the same containers and verifiers as §4.2, so training consumes the verifier's native reward rather than a learned proxy. Figure 14 shows the three phases of an iteration, multi-turn rollout, verifier reward evaluation, and one optimizer step, with weights updated asynchronously. Rollout and training workers each own whole GPUs, while the agent-loop workers that drive episode execution share CPU cores or claim both CPUs and GPUs, and every node's devices join one pool per device class. Episodes are expensive and uneven in length, since one may rebuild a Fortran solver and run a full simulation, so a synchronous loop would wait for the slowest episode of every batch. Giving rollout and training disjoint GPUs lets generation continue against the current weights while the optimizer runs; on a representative step the update took 2,069 of 3,210 seconds and rollout 751 seconds. Agent-loop workers are light CPU processes, so many episodes execute concurrently on shared cores (Appendix H.2).

Objective. For a prompt group of $G = 8$ trajectories with rewards R_i, the group-relative advantage is given by Eq. (2), which omits the division by the group standard deviation of the original formulation, following Dr. GRPO (Liu et al., 2025), because near-bimodal group rewards would otherwise inflate isolated successes (Appendix H.2). Tokens are then updated with a PPO-style clipped ratio whose bound is asymmetric ($\varepsilon_{\text{low}} = 0.2$, $\varepsilon_{\text{high}} = 0.3$), so that rarely sampled repair actions retain room to gain probability mass (Yu et al., 2025); the loss is aggregated by token mean; Eq. (4) and the remaining settings, unchanged across both runs, are in Appendix H.2.

Budget cutoffs are not failed repairs: a long-horizon challenge in science RL. In long-horizon scientific tasks, an episode can end because the repair finishes, the turn cap is exhausted, or the response budget is exhausted. Only the first indicates whether the repair succeeded, yet outcome-only reward scores all three alike. A budget-truncated episode therefore takes $\hat{A}_i < 0$ from Eq. (2), suppressing every token it generated. With token-mean aggregation, this penalty is especially strong for the very trajectories that consume the most tokens. The objective consequently creates a shortcut: the policy can improve either by solving the scientific task or by simply making its trajectories shorter.

The effect is especially harmful because the longest trajectories are often the ones attempting the hardest repairs. Penalizing them for exhausting a budget therefore discourages the policy from

exploring solutions that require extended interaction with the scientific environment. We observe this failure mode directly in an unmasked run. Reward initially improves but then collapses below its starting point, while tokens per turn fall by more than a factor of three. Shorter turns cause more trajectories to exhaust the turn cap, which increases truncation and reinforces the same pathological signal. The model can thus enter a self-reinforcing cycle in which shortening the trajectory looks preferable to making progress on the repair.

The fix is to separate the role of a truncated trajectory as a reward observation from its role in policy optimization. A harness cutoff says nothing about whether the tokens generated before the cutoff were useful, so those tokens should not receive a policy gradient. At the same time, the trajectory's reward remains informative for the group baseline. We therefore keep budget-truncated trajectories in the group baseline and mask them out of the loss (Eq. (5) in Appendix H.2). Its reward still enters Eq. (2), so completed episodes gain a positive advantage relative to truncated ones. By the same reasoning, we switch off DAPO's reward-side length penalty. Length here reflects the difficulty of localizing and repairing a defect, not a quantity to penalize (paired comparison and penalty audit in Appendix H.2).

Online verifier feedback substantially improves held-out scientific reward. With the truncation mask in place, held-out reward rises in both environments (Figure 6, right column): after 30 training steps, LAPS from 0.357 to 0.857 ($2.4\times$) and MITgcm-biogeo from 0.286 to 0.571 ($2.0\times$) over the base Qwen3.5-4B checkpoint.

The training dynamics explain where these gains come from (Figure 6). On LAPS, mean training reward increases from 0.427 to 0.828, while the fraction of completed episodes rises from 0.683 to 0.883. At the same time, budget truncation drops from 39.5% to 6.6%. MITgcm-biogeo shows the same trend, with reward increasing from 0.381 to 0.597, the completion rate increasing from 0.571 to 0.747, and truncation decreasing from 34.1% to 23.8%. The policy increasingly converts long-running attempts that previously died at a budget cutoff into episodes that reach a verifier verdict and receive meaningful feedback.

The reduction in truncation does not come from shorter responses. On LAPS, average tokens per turn increase from 1103 to 1135, and on MITgcm-biogeo they increase from 780 to 914. The policy therefore becomes substantially more successful without collapsing its interaction horizon.

The two environments differ in scientific domain, codebase, task bank, and defect families, yet both show higher training reward, lower truncation, and better held-out reward under the same procedure. Direct verifier feedback therefore trains scientific repair across simulators once the objective handles long-horizon execution.

H Learning settings and supplementary results

H.1 Supervised fine-tuning

Qwen3.5-4B, Qwen3.5-9B, and Qwen2.5-72B-Instruct were trained with ms-swift on GPT-5.6-sol demonstrations selected by numerical-equivalence verification. The task-identifier-disjoint corpus contains 4,567 training segments from 564 tasks and 544 validation segments from 81 tasks. The loss supervises new assistant actions and tool calls; instructions, observations, reused context, and flagged repetitive or failed actions are masked. Related variants were not independently audited for overlap.

Example construction. Each example consists of the initial instruction, bounded interaction history, and a new target segment ending at a complete assistant–observation block. The loss supervises selected assistant tokens, including tool calls, in the new segment. Instructions, observations, and reused history are masked. Tool-action targets identified as failed, repetitive, or part of an extended read-only sequence are also masked, with their messages retained as context. Successive examples can thus share context without repeating supervised targets.

LoRA targets all linear layers with rank 32, scaling 64, and dropout 0.05. Runs use BF16, global batch size 64, a 36,864-token sequence limit, learning rate 2×10^{-5}, 3% warmup, cosine decay, and three epochs (216 steps). Final checkpoints are evaluated without selecting them on public-

benchmark performance. Table 2 reports teacher-forced validation loss and token accuracy on unmasked targets.

Table 2: **SFT validation diagnostics.** Loss is measured after epochs one and three; final token accuracy uses unmasked validation targets.

Model	Loss, epoch 1	Loss, epoch 3	Final acc. (%)
Qwen3.5-4B	0.383	0.340	91.12
Qwen3.5-9B	0.369	0.325	91.33
Qwen2.5-72B-Instruct	0.401	0.313	92.01

Evaluation and confirmation. The three-model study screens 33 benchmark configurations with 20–128 items each. Multiple-choice configurations use continuation likelihoods or option extraction from generated responses. Other generated answers are scored by numerical or symbolic comparison, exact match, or execution against frozen tests. Generation is greedy, with thinking disabled and a 4,096-token output limit. Within each model pair, both checkpoints use the initial model's tokenizer and the same context limit: 65,536 tokens for 4B/9B and 32,768 for 72B. Selected findings are evaluated on disjoint confirmation items. A separate 4B/9B extension covers 21 further configurations. Reported intervals are paired-bootstrap intervals without correction for multiple comparisons.

Screening gains by benchmark. Beyond the largest code gains reported in §4.3, the remaining code comparisons show smaller gains. HumanEvalFix Rust improves by 3.66 points for 9B. For 72B, APPS Introductory improves by 6.25 points and LiveCodeBench Execution by 5.47 points, while HumanEvalFix C++ and HumanEval gain 4.69 and 3.125 points, respectively. On QuixBugs Python, both 4B and 72B improve by 5.0 points, from 0.600 to 0.650 and from 0.900 to 0.950, respectively. The same absolute gain therefore occurs at different initial performance levels.

The improvements also extend to reasoning and knowledge benchmarks. For 9B, BBH Word Sorting increases from 0.240 to 0.576 (+33.6 points), alongside a 3.125-point gain on GSM8K. For 4B, BBH multistep arithmetic increases from 0.920 to 0.968 (+4.8 points), and MMLU-Pro Computer Science gains 4.69 points. ARC-Easy improves by 3.125 points for 72B.

Table 3 retains the disjoint confirmation comparisons.

Table 3: Confirmation comparisons shown in Figure 5(b). Scores are percentages; changes and 95% paired-bootstrap intervals are percentage points. Rows retain their own accuracy, exact-match, or execution metric. Intervals are unadjusted for multiple comparisons.

Dataset / model (n)	Initial	SFT	Change	95% interval
CodeXGLUE defect / 4B (2,604)	45.93	52.92	+6.99	[4.03, 9.87]
CodeXGLUE refinement / 9B (5,707)	1.17	2.37	+1.19	[0.89, 1.51]
CodeXGLUE refinement / 72B (5,707)	1.63	2.40	+0.77	[0.53, 1.03]
BBH Word Sorting / 9B (125)	27.20	63.20	+36.00	[26.40, 46.40]
BBH arithmetic / 4B (125)	92.80	98.40	+5.60	[0.80, 11.20]
HumanEvalFix Python / 9B (36)	86.11	77.78	-8.33	[-19.44, 2.78]

On the 250-item BBH full splits, 9B Word Sorting improves from 25.6% to 60.4% and 4B multistep arithmetic from 92.4% to 97.6%. Their respective 125-item confirmation comparisons are 27.2% to 63.2% and 92.8% to 98.4%. Full splits combine screening and confirmation and are not additional independent replications. The 72B HumanEvalFix C++ screening gain does not persist on confirmation. Machine-readable confirmation and screening records are retained in the accompanying data files; Figure 5 does not summarize all benchmarks as uniformly improved.

H.2 Reinforcement learning

RL starts from base Qwen3.5-4B without SFT initialization. LAPS uses 99 tasks split 85/14. The MITgcm-biogeo run records 87 tasks with a source split of 64/23, while its logged validation reads

21 tasks. A rebuilt split is 66/21 and is not substituted for the reported run. Factory splits group defect-family and source-tree variants. Both training and validation use L1 hints identifying file, line, and edit class without the repair, and the held-out comparison is within each environment.

Execution stack. Every iteration rolls out episodes, grades them, and takes one optimizer step (Figure 14 in §4.4). Generation runs in vLLM, optimization in PSRL, a customization of the veRL trainer that drives a black-box harness, and episodes execute in harbor through the same Terminus-2 containers and verifiers as the evaluation campaign. This design lets training consume the verifier's native reward rather than a learned proxy, since the trainer does not need to model the execution inside each container. The resulting episodes are nevertheless expensive and variable in length. Rollout and training therefore occupy disjoint GPUs and run concurrently, allowing the training workers to make progress while new episodes are being evaluated.

Optimization and execution. The algorithmic choices are motivated in §4.4. This section records the settings behind them and the run-level evidence for the truncation mask. Episode execution uses `TruncatingTerminus2` with 16 concurrent episodes, a 2,700-second task timeout, a 900-second verifier timeout, and 4,096-byte observation truncation. Each run uses 24 H20 GPUs with 95GB per device: eight for generation (tensor parallelism 2, four instances) and sixteen for training (FSDP2, sequence parallelism 4). On a representative step the optimizer update took 2,069 of 3,210 seconds and rollout 751 seconds, which is the utilization motivating disjoint rollout and training pools. Trajectory capture is token-in-token-out: each turn's message prefix is hashed so that turn $k+1$ extends turn k, and reasoning is retained across turns so every turn is a strict prefix extension. A stock chat template that strips the previous turn's reasoning would miss the prefix hash and reopen a trajectory per turn, making storage and backward cost grow with the square of the turn count. Table 4 records the optimization settings.

Table 4: Reinforcement-learning run settings, carried unchanged by both reported runs. The left block lists the algorithmic choices motivated in §4.4, the right block the remaining numerical settings. The advantage baseline follows Dr. GRPO (Liu et al., 2025), the asymmetric clip follows DAPO Clip-Higher (Yu et al., 2025), and rollout correction is token-level truncated importance sampling.

Algorithmic choice	Value	Run setting	Value
Advantage baseline	Group mean, no std.	Batch / rollouts	16 / 8
Clipping	0.2 / 0.3	Learning rate	10^{-6}
Loss aggregation	Token mean	Warmup / decay	3 steps / 0.1
Truncated episodes	Masked from loss	Turn limit	50
Length penalty	Off	Prompt / response	2,048 / 65,536
Rollout correction	Cap 2.0	Temperature	1.0
Dynamic sampling	Off	Staleness	1 update
KL / entropy bonus	Off / off	Measured KL	6×10^{-4}
		Generation / trainer	vLLM / veRL

The advantage of Eq. (2) omits the standard-deviation normalization of the original GRPO formulation. Verifier rewards in our setting are close to bimodal, with most groups scoring nearly all zero or nearly all one. Dividing by a small standard deviation would therefore turn an isolated success into an disproportionately large advantage. With this advantage, each token of trajectory i contributes

$$\ell_{i,t}(\theta) = \min\big(r_{i,t}\hat{A}_i,\ \mathrm{clip}(r_{i,t},\, 1-\varepsilon_{\mathrm{low}},\, 1+\varepsilon_{\mathrm{high}})\,\hat{A}_i\big), \tag{4}$$

where $r_{i,t}$ is the per-token probability ratio against the sampling policy. The bound is asymmetric, with $\varepsilon_{\mathrm{low}} = 0.2$ and $\varepsilon_{\mathrm{high}} = 0.3$, so that rarely sampled repair actions retain room to gain probability mass (Yu et al., 2025), and the loss is aggregated by token mean. Rollout correction uses token-level truncated importance sampling to account for policy staleness, which arises because a step trains on trajectories from the previous weights and because rollout runs in BFloat16 while training uses higher precision. The weight is $w_{i,t} = \min(\exp \rho_{i,t},\, C)$ with $C = 2.0$, where $\rho_{i,t}$ is the trainer-minus-rollout log-probability difference for that token, computed in log space. SFT and RL are reported as independent studies.

Termination handling and diagnostics. Budget-truncated trajectories retain their reward contribution to the group baseline but have no token-loss contribution:

$$m_{i,t} = \begin{cases} 0 & \text{if } \tau_i \in \mathcal{T}_{\text{budget}}, \\ \mathbb{1}[\text{token } t \text{ is model-generated}] & \text{otherwise,} \end{cases} \tag{5}$$

The implementation applies the mask after termination classification, so $\mathcal{T}_{\text{budget}}$ contains turn- and response-budget cutoffs and excludes infrastructure timeouts and completed grading errors. Prompt tokens and harness padding are excluded from the mask. A batch in which every trajectory is budget-truncated would give the token-mean denominator no support, so it is clamped to at least one and such a batch yields a zero gradient.

The run-level comparison behind that choice is Table 5. Across the collapsed run, effective sample size remains in $[0.950, 0.978]$ and rollout/trainer log-probability correlation recovers from 0.938 to 0.955 while reward falls, which rules out importance-sampling degradation as the cause. Over the reported runs, effective sample size stays in $[0.950, 0.978]$ for LAPS and $[0.953, 0.976]$ for MITgcm-biogeo, with log-probability correlations in $[0.948, 0.984]$ and $[0.965, 0.977]$. These bound weight transfer and policy mismatch. The audit of the reward-side penalty we switch off found it firing on 28% of samples over one run, of which 119 of 142 were already-failing cases, and hitting finished episodes scoring 1.0 about 64% of the time. The fraction of groups with zero reward variance, which contribute no gradient under Eq. (2), grows from 10.0% to 47.5% over the LAPS run and from 27.5% to 41.3% over MITgcm-biogeo, comparing the first and last five recorded steps.

Table 5: Run-level comparison with and without the truncation mask, from the first to the last recorded step of each run, with the unmasked run shown through its peak to its collapse. Tokens per turn and turns per episode decompose response length.

	Unmasked	Masked
Mean reward	$0.339 \to 0.573 \to 0.078$	$0.487 \to 0.914$
Tokens per turn	$1125 \to 327$	$1095 \to 1165$ (flat)
Truncation rate	$29\% \to 39\%$	$28.9\% \to 2.3\%$
Turns per episode	$34.9 \to 49.5$	$32.2 \to 24.1$
Entropy	Rising	$0.586 \to 0.348$

Comparing the first and last five recorded steps of each reported run, training reward rises from 0.427 to 0.828 on LAPS and from 0.381 to 0.597 on MITgcm-biogeo, budget truncation falls from 39.5% to 6.6% and from 34.1% to 23.8%, and entropy falls from 0.602 to 0.381 and from 0.549 to 0.313. Turns per episode fall from 34.7 to 24.5 on LAPS while tokens per turn hold near 1100, so the shortening reflects earlier completion instead of truncated turns.

I DIFFICULTY: POLICY, BUDGET, AND DESIGN HISTORY

Difficulty is conditional on the solver and the routes by which it can recover a solution. Design labels express intent; scientific validity and measured hardness require different evidence. A zero score can reflect solver failure, budget exhaustion, reward misalignment, or an invalid task. No model-based diagnosis can override deterministic validity anchors.

The design history illustrates why removing an obvious shortcut need not create harder scientific work. Literal-factor and no-new-literal repairs remain solvable through source reading and check-implied localization. Coupled $k = 2$ defects remove partial-repair feedback: Astra solves 3/8, Opus 2/8, and both fail on 5/8 tasks with one attempt per model. Single-routine excisions remain reconstructible: Astra solves 4/4 and Opus 2/4. These observations motivate factory refinement, not general difficulty laws: each cheap localization channel that an agent can exploit, such as file timestamps, reference hashes reachable in the image, or upstream unit tests that cover the defect site, is now a gate at authoring time, and recall of public code is recorded as a task property rather than assumed away. The complete design-history table and its historical timestamp and retrieval caveats follow in Table 6.

The historical design rounds below retain their original completed-trial denominators and distinguish observed solution channels from causal explanations.

Table 6: **Cheap channels limit intended repair difficulty.** One attempt per task/model; early rows retain completed audit snapshots. A = gpt-6-astra; O = claude-opus-5. Rates are solved/strict-valid completed trials; invalid and pending trials are excluded. Final coupled/excision totals follow the campaign report; expansion reports bank yield. Channels describe observed routes, not causal attribution.

Round / design	Solves / yield	Dominant channel or failure pattern
1: literal factors	A+O: 28/28	Source reading; numeric grep returned the defect later in 11/13 A solves.
2: no new literals	A+O: 26/26	Check-implied reads (A: 9/13); mtime-first (A: 3/13).
2: normalized	A+O: 28/28	Module reads and call chains; no mtime-first localization.
3: cross-module	A: 8/8; O: 5/8	Feature-symbol search and slice reading; O times out after exposure.
4: coupled $k = 2$	A: 3/8; O: 2/8; both fail: 5/8	No partial-repair feedback: agents self-test one repair and stop or revert it; every solved case had a site exposed by run logs or upstream tests.
5: excisions	A: 4/4; O: 2/4	A re-derived 30–60-line routines in 6–7 min from call sites and conventions; single-routine excision is insufficient.
6: expansion	13 hard / 366 candidates	53 further environments; 7 coupled and 6 excision tasks banked by design and probes, without new model difficulty measurements.

Early trials retain the timestamp/hash caveats in §5.

J ENVIRONMENT SOURCES

The frozen registry cohort contains 27 upstream scientific codebases and 64 derived environments. The list retains every source, its primary citation, the upstream software identifier, and the license recorded in the codebase report; the environment-to-source map and pinned repository records are designated for anonymous supplementary material.

Athena++ astrophysical radiation hydrodynamics and MHD (Stone et al., 2020). `athena`, BSD-3-Clause.
Basilisk spacecraft and balanced reaction-wheel dynamics (Kenneally et al., 2020). `basilisk`, ISC.
dscribe (Himanen et al., 2020). `dscribe`, Apache-2.0.
EDKit.jl (Ren, 2026). `EDKit.jl`, MIT.
eftcamb (Hu et al., 2014). `eftcamb`, GPL-3.0 (EFTCAMB part); CAMB licence for unmodified CAMB.
EPOCH particle-in-cell code (Arber et al., 2015). `epoch`, GPL-3.0 (task metadata); GPL-3.0-or-later in inspected Fortran file headers.
G4CMP (Kelsey et al., 2023). `G4CMP.git`, GPL-3.0-or-later (bundles Geant4 licence and Qhull).
Gkeyll Computational Plasma Physics Package (Hakim & Gkeyll Team, 2026). `gkeyll`, MIT.
LAPS: 3D and 2D compressible Hall-MHD (Shi et al., 2024). `LAPS`, GPL-2.0.
Meep 1.34.0 — finite-difference time-domain electromagnetics (Oskooi et al., 2010). `meep`, GPL-2.0-or-later.
Mink differential inverse kinematics (Zakka, 2026). `mink`, Apache-2.0.
MITgcm (MIT General Circulation Model) (Marshall et al., 1997). `MITgcm`, MIT.
NEST: Noble Element Simulation Technique (Szydagis et al., 2011). `nest`, Apache-2.0.
EPREM (Schwadron et al., 2010). `eprem.git`, GPL-3.0-only.
Phantom (Price et al., 2018). `phantom`, GPL-3.0.
PLUTO — Godunov-type computational astrophysical fluid dynamics (Mignone et al., 2007). `pluto`, GPL-2.0.
PyAMG (Bell et al., 2023). `pyamg`, MIT.
pymatgen (Ong et al., 2013). `pymatgen`, MIT.
pymatgen-core (Ong et al., 2013). `pymatgen-core`, MIT.
pyXSIM (ZuHone et al., 2014). `pyxsim`, BSD-3-Clause.
QuTiP — Quantum Toolbox in Python (Lambert et al., 2026). `qutip`, BSD-3-Clause.
S4: Stanford Stratified Structure Solver (Liu & Fan, 2012). `S4`, GPL-2.0-or-later.
Scirpy — single-cell immune receptor repertoire analysis (Sturm et al., 2020). `scirpy`, BSD-3-Clause.
NASA SimuPy Flight: NESC 6-DoF dynamics (Margolis & Lyons, 2022). `simupy-flight`, NASA-1.3.
Stim — a fast stabilizer circuit simulator (Gidney, 2021). `Stim`, Apache-2.0.
strax: streaming analysis for xenon TPCs (Aalbers et al., 2026). `strax`, BSD-3-Clause.
TSID TALOS fixed-contact whole-body inverse dynamics (Del Prete et al., 2016). `tsid`, BSD-2-Clause.