

D3-Gym: Constructing Real-World Verifiable Environments for Data-Driven Discovery

Hanane Nour Moussa^{1*}, Yifei Li^{1*}, Zhuoyang Li¹, Yankai Yang², Cheng Tang¹,
Tianshu Zhang¹, Nesreen K. Ahmed³, Ali Payani³, Ziru Chen¹, Huan Sun¹

¹The Ohio State University

²University of Wisconsin-Madison

³Cisco Research

{moussa.45, li.14042, sun.397}@osu.edu

Abstract

Despite recent progress in language models and agents for scientific data-driven discovery, advancing their capabilities is held back by the absence of verifiable environments representing real-world scientific tasks. To fill this gap, we introduce D3-Gym, the first dataset of automatically constructed *verifiable environments* for scientific Data-Driven Discovery. D3-Gym comprises 565 tasks from 239 real scientific repositories across four disciplines, each with a natural language instruction, an executable environment with pre-installed dependencies, dataset previews, a reference solution, and an *automatically synthesized* evaluation script. Our evaluation scripts achieve 87.5% agreement with human-annotated gold standards and strong alignment in domain-specific evaluation logic. Training on trajectories sampled from D3-Gym yields consistent gains across Qwen3 models on ScienceAgentBench, boosting Qwen3-32B SR@3 by 7.8 absolute points and shrinking the gap with strong proprietary models. We further illustrate, through case studies, how D3-Gym environments can serve as a testbed for studying agentic optimization loops such as Autoresearch on real scientific workflows. We open-source D3-Gym, its creation workflow, sampled trajectories, and training scripts at <https://github.com/OSU-NLP-Group/D3-Gym>.

1 Introduction

Language agents (Su et al., 2024; Sumers et al., 2024) are reshaping scientific research. Powered by large language models (LLMs), they can now meaningfully assist researchers in their workflows, from literature synthesis (Skarlinski et al., 2024; Asai et al., 2026) to domain-specific scientific reasoning (Baker et al., 2025; Narayanan et al., 2025). In particular, there has been significant research interest in developing language agents for data-driven

*Equal contribution.

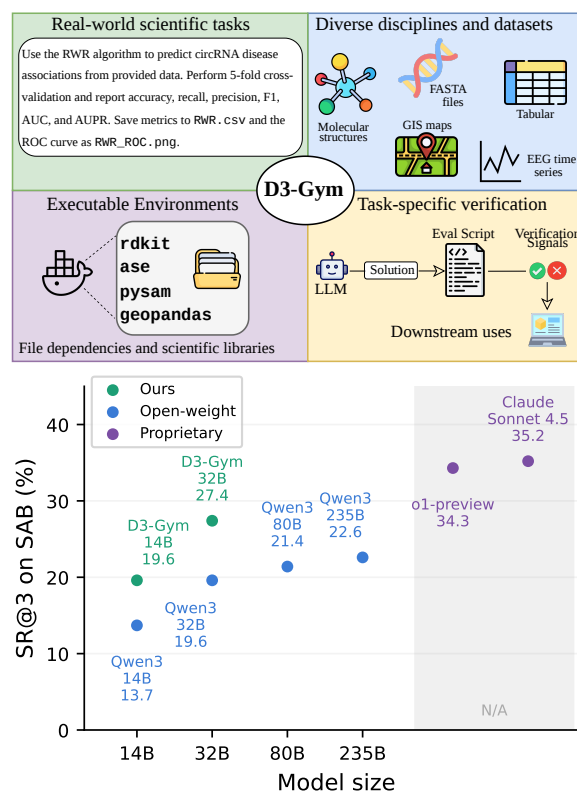


Figure 1: Top: Overview of D3-Gym. Bottom: Success Rate (SR@3) on ScienceAgentBench (SAB) (Chen et al., 2025) of D3-Gym trained models based on Qwen3-14B and Qwen3-32B and other larger open-weight and proprietary reasoning models.

discovery (Hey et al., 2009; Majumder et al., 2024), where agents can programmatically test new hypotheses (Majumder et al., 2025; Mitchener et al., 2025) and derive scientific insights from data (Chen et al., 2025; Shojaee et al., 2025a,b). Despite this progress, further advancing these capabilities, particularly for open-weight LLMs (Zhu et al., 2025), is held back by the absence of suitable infrastructure for scientific tasks.

A critical component in this infrastructure is *real-world verifiable environments*. Recent work such as Autoresearch (Karpathy, 2026) highlights their value: given a manually prepared task instruction, a dataset, an evaluation script, and a repository

with pre-installed dependencies, coding agents can use the executable feedback loop to improve performance and run dozens of experiments in a few hours. However, constructing such environments requires substantial manual effort, making them difficult to scale beyond a small number of tasks.

In this paper, we fill this gap with an automated pipeline that constructs real-world, verifiable coding environments for data-driven discovery. Unlike software engineering, where unit tests in repositories can be directly repurposed as verification signals (Yang et al., 2025b; Jain et al., 2025; Pan et al., 2025), automatically constructing environments for scientific tasks is particularly challenging because related repositories rarely contain pre-existing evaluation scripts, and programs produce domain-specific outputs whose correctness cannot be assessed by any universal criterion. To address this, we start from a large pool of candidate tasks collected using AutoSDT (Li et al., 2025) and design a series of rigorous filtering steps, followed by task execution and output validation, to retain high-quality, ecologically valid task instances. We then synthesize evaluation scripts in two stages: first writing a detailed, task-specific evaluation plan, and then translating that plan into an executable script to complete environment construction.

The resulting dataset, D3-Gym, comprises 565 verifiable environments from 239 scientific repositories in bioinformatics, computational chemistry, geographic information science, and psychology and cognitive neuroscience (Figure 1). Each instance includes an executable environment with pre-installed dependencies, input data files and previews, a natural language task instruction, a reference solution adapted from the original repository, and a task-specific evaluation script with metrics and acceptance criteria. To our knowledge, D3-Gym is the first automatically constructed dataset with verifiable environments for scientific data-driven discovery.

Through comprehensive experiments, we demonstrate the quality of synthesized evaluation scripts and the utility of D3-Gym in two complementary settings. On 50 tasks with human-annotated gold evaluation scripts, our synthesized evaluation scripts achieve 87.5% agreement on pass/fail verdicts and closely match the underlying evaluation logic. Furthermore, rejection sampling fine-tuning (Yuan et al., 2023) on D3-Gym substantially improves Qwen3 models (Qwen Team, 2025) on ScienceAgentBench (Chen

et al., 2025): our 32B model improves SR@3 by 7.8 absolute points, surpasses Qwen3-235B-A22B, and approaches strong proprietary models such as OpenAI o1 (Figure 1). Additionally, two case studies pairing a coding agent with D3-Gym environments in an Autoresearch-style search loop illustrate how the synthesized verification signals can support iterative agentic refinement and surface concrete behaviors of frontier coding agents on real scientific workflows. Overall, D3-Gym provides a foundation for training, evaluation, and iterative refinement for data-driven discovery in real-world, verifiable environments.

2 Constructing Verifiable Environments

As shown in Figure 2, we construct D3-Gym through a four-stage workflow. The first three stages (§2.1) collect candidate tasks from scientific repositories, filter them for data integrity, and execute them to obtain reference outputs. These stages ensure task quality and produce the essential context that enables the core contribution of our workflow: automated generation of task-specific evaluation scripts with scientifically grounded metrics and acceptance criteria (§2.2). We validate these scripts against human-annotated gold standards in §2.3. Further details are in Appendix A.

2.1 Task Pre-processing

Candidate Task Collection. We collect an initial pool of candidate tasks using AutoSDT (Li et al., 2025), which crawls scientific repositories from GitHub, applies multi-step filtering to identify files implementing data-driven scientific workflows, adapts them into reference code solutions, and pairs each with a natural language task instruction. The pipeline also extracts a dependency folder for each task with all file dependencies from the original repository. Expert evaluation confirms that 93% of collected tasks are scientifically meaningful (Li et al., 2025). We ensure that all repositories used in ScienceAgentBench (Chen et al., 2025) are *excluded*, as it is our evaluation set.

Filtering and Dataset Preview Creation. A limitation of AutoSDT is that its code adaptation stage may substitute synthetic data when an input file is absent from the original repository (Li et al., 2025). Since our goal is to construct high-fidelity environments grounded in real scientific workflows, we discard such tasks with synthetic data and retain only those whose reference solutions operate exclu-

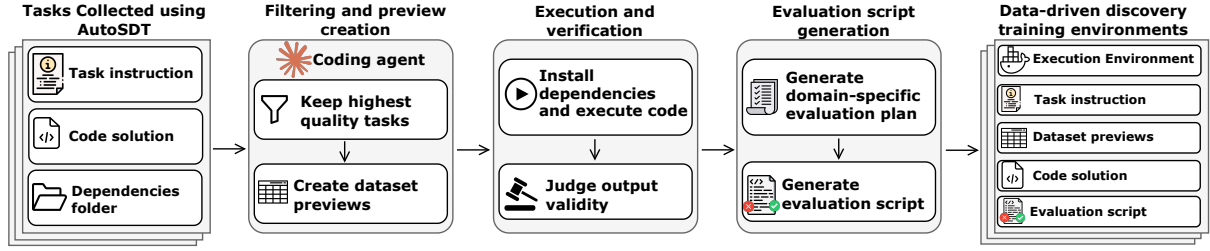


Figure 2: Overview of the D3-Gym construction workflow. Candidate tasks from scientific repositories (Li et al., 2025) undergo quality filtering and dataset preview creation, then execution and validation to obtain reference outputs. These provide the context needed to generate high-quality, task-specific evaluation scripts via a planning-then-coding approach.

sively on real data files in the original repository. Claude Code (Anthropic, 2025a) verifies file dependencies and, for passing tasks, produces a *dataset preview file* for each input data file: a structured excerpt exposing the data schema. These previews provide concrete data context and later inform evaluation script generation.

Execution and Output Verification. We execute each remaining reference solution in an isolated environment, using `pipreqs`¹ to install required packages. While the previous step ensures data integrity, execution can still fail or produce degenerate outputs due to environment mismatches or subtle errors from code adaptation. We employ a multi-modal LLM-as-judge (GPT-5.2, 92.31% agreement with human; Appendix A.3) to verify all requested outputs were produced and each output is meaningful and non-degenerate. Tasks failing either criterion are discarded. The verified outputs, together with the dataset previews, form the context for evaluation script generation.

2.2 Evaluation Script Generation

A central challenge is generating discriminative evaluation scripts that correctly capture what constitutes a correct solution for a given task while rejecting incorrect ones. Inspired by Saha et al. (2025) and Yang et al. (2025a), we decompose evaluation script generation into two phases: *planning* and *coding*. This separation encourages the planning LM to reason carefully about scientific validity without being distracted by implementation details, while allowing the coding LM to focus on producing correct, executable code faithful to the plan. This design choice is validated by our ablation in §2.3, where removing the planning phase significantly reduces performance.

In the planning phase, we provide Claude Sonnet

4.5 with the task instruction, dataset previews, and verified reference outputs and leverage its parametric scientific knowledge to produce a detailed, task-specific evaluation plan. The plan specifies which output artifacts to inspect, which metrics are scientifically appropriate for each artifact, and the acceptance criteria, including tolerance thresholds, exact versus approximate matching, and domain-justified performance bounds. The evaluation plan is passed to Claude Sonnet 4.5 in a separate call, which generates the final executable evaluation script in a single pass. We refer to these automatically generated scripts as *silver evaluation scripts*, to distinguish them from human-annotated *gold evaluation scripts* used to assess their reliability.

2.3 Validating Evaluation Script Quality

A critical property of any evaluation script is the reliability of its verification signal. We therefore rigorously evaluate our silver scripts along two complementary dimensions: (1) *execution-based agreement*, which measures whether silver scripts produce the same pass/fail verdicts as human-annotated gold scripts, and (2) *evaluation logic agreement*, which assesses whether the methodology aligns with that of gold scripts.

Setup. We curate a held-out set of 50 task instances with gold evaluation scripts manually written and validated by Ph.D. students over 175 person-hours. For each, we generate a silver evaluation script using the method in §2.2. To obtain candidate solutions for execution-based metrics, we sample 10 solutions per task from Claude Opus 4.6 and GPT-5.4. These models were chosen because their higher execution success rates yield the largest pool of solutions that proceed to evaluation. After filtering for successful execution, this produces 424 candidate solutions, each scored by both the silver and gold scripts.

¹<https://pypi.org/project/pipreqs/>

Method	Pass/Fail Agreement (%)			Evaluation Logic (1–5, ↑)		
	Acc.	Spec.	Recall	Metric	Threshold	Artifact
Direct Prompting	86.1	100.0	0.0	2.07	2.16	2.66
Silver Eval. Scripts (Ours)	87.5	91.0	66.1	4.00	3.82	4.20
w/o Planning	74.8	86.3	3.4	2.00	2.04	2.68
w/o Dataset Preview	63.2	70.4	18.6	1.96	1.74	2.64
w/o Code Output	85.1	98.9	0.0	2.89	2.64	2.62

Table 1: Quality validation of silver evaluation scripts against human-annotated gold scripts. *Left*: pass/fail agreement. High specificity with near-zero recall (e.g., Direct Prompting) indicates scripts that reject nearly all solutions indiscriminately. *Right*: evaluation logic alignment on metric choice, threshold & tolerance, and target artifact.

Metrics. (1) **Accuracy** measures the proportion of candidate solutions for which the silver and gold scripts return the same verdict. (2) **Specificity** measures the fraction of gold-failing solutions that also fail the silver script, the property that ensures incorrect solutions do not slip through verification into downstream use. (3) **Recall** measures the fraction of gold-passing solutions that also pass the silver script, capturing the rate at which correct solutions are retained. Further, we assess silver scripts’ scientific validity via an LLM-as-judge (85% exact, 98% within-1 agreement with human annotators; Appendix A.3) that scores each silver script against its gold counterpart on three aspects using a 1–5 Likert scale: (4) **Metric Choice**, (5) **Threshold & Tolerance**, and (6) **Target Artifact**. Setup details are in Appendix B.

Baselines. We compare our full method (§2.2) against a direct prompting baseline that provides the task instruction and code solution to an LLM and produces the evaluation script in one step. We also ablate three components: evaluation planning, dataset preview access, and code output access. All evaluation script generation methods use Claude Sonnet 4.5 as the backbone LLM.

Results. As shown in Table 1, our pipeline is the only configuration that achieves substantial recall (66.1%) while preserving high specificity (91.0%). The baseline and all ablations show high specificity and low recall, indicating scripts that reject solutions indiscriminately. Removing evaluation planning reduces recall to just 3.4%, removing the dataset preview introduces schema expectation errors, and removing access to code output reduces recall to 0%. Our method consequently achieves the highest accuracy overall, with silver-accepted solutions reliably corresponding to those gold scripts would accept and a substantial fraction of correct solutions retained; the remaining disagreements

concentrate at the gold pass/fail boundary, where solutions narrowly satisfy gold thresholds but fall outside the silver criteria (Appendix B.4). The evaluation logic agreement scores further confirm silver scripts’ methodological soundness: our method substantially outperforms the baseline and all ablations across metric choice, threshold & tolerance, and target artifact.

3 D3-Gym

Using our verifiable environment creation workflow (§2), we present D3-Gym. Each task instance in D3-Gym comprises an executable environment with pre-installed dependencies, dataset preview files, a natural language task instruction, a scientist-authored reference solution, and a task-specific evaluation script generated automatically. D3-Gym comprises a diverse set of 565 tasks drawn from 239 scientific repositories. The synthesis of each task costs around \$3. We report the per-stage environments synthesis cost in Appendix B.2.

Discipline, modality, and task type distribution. As shown in Figure 3, D3-Gym spans four scientific disciplines: Computational Chemistry, Bioinformatics, Geographic Information Science, and Psychology & Cognitive Neuroscience. Task types reflect the full arc of real-world scientific workflows, ranging from statistical analysis, numerical computation, and visualization to specialized tasks such as feature engineering, machine learning, and signal and image processing. D3-Gym also encompasses a wide spectrum of scientific data formats: in addition to conventional tabular, textual, and image data, many tasks operate on domain-specific representations such as molecular structure encodings (e.g., SMILES, FASTA), molecular simulation trajectories (e.g., TRAJ), and geospatial records (e.g., GPS logs). Outputs are similarly varied, including visualizations and serialized artifacts

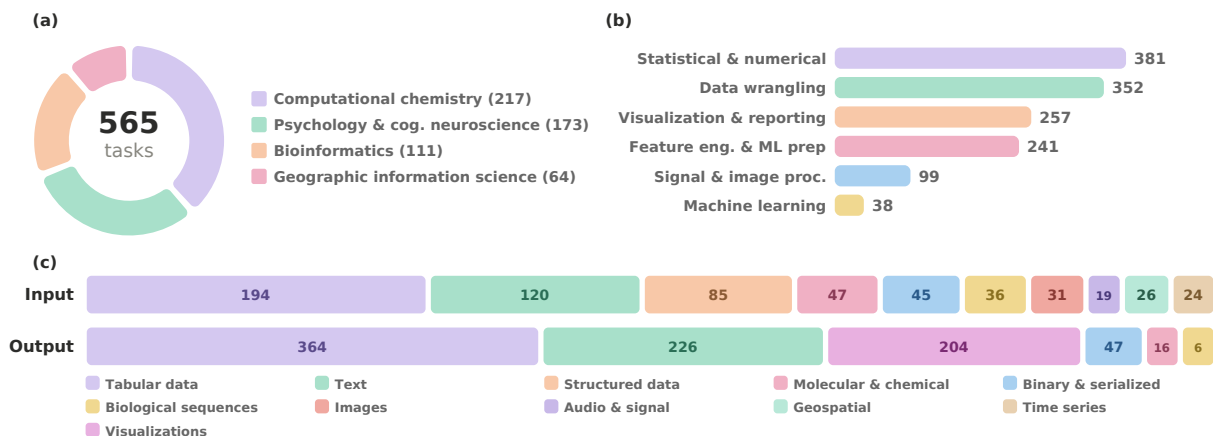


Figure 3: D3-Gym statistics. (a) Distribution of tasks by scientific discipline. (b) Distribution of tasks by type. Tasks carry multiple type labels (mean 2.4 labels per task). (c) Distribution of tasks by input and output modality. Tasks may carry multiple modalities (input: mean 1.11, max 3; output: mean 1.53, max 4).

such as trained models, NumPy arrays, time series, maps, and domain-specific structures. We provide examples in Appendix B.1.

Scientific package coverage and task difficulty.

Tasks draw on both the general-purpose scientific stack (numpy, pandas, matplotlib, scipy) and domain-specific libraries including rdkit, ase, Bio, pysam, LFPy, dyconnmap, geopandas, and folium. This breadth of domain-specific tooling reflects D3-Gym’s genuine scientific diversity and is a key source of its difficulty. To gauge task difficulty, we sample 75 tasks and evaluate GPT-5.4, Claude Opus 4.6, Qwen3-235B-A22B, and Qwen3-32B (Appendix B.5). Even the most capable frontier models solve only around a third of the tasks (GPT 5.4: 38.66%), and none consistently produce fully executable code, with GPT-5.4 and Claude Opus 4.6 achieving only 82.66% and 78.66% valid execution rates. Overall, these results confirm our tasks are solvable yet non-trivial.

4 Training using D3-Gym

D3-Gym provides a versatile training environment that supports various approaches for improving model capabilities. Here, we demonstrate its utility by training open-weight models for scientific data-driven discovery—a setting where reliable, transparent, and reproducible systems are especially important (Spirling, 2023). Given the current performance gap of open-weight models on these tasks, we use training approaches that provide dense supervision and are effective at expanding model capabilities (Yue et al., 2025). We therefore adopt rejection-sampling fine-tuning (RFT)

(Yuan et al., 2023), where a model generates trajectories with full reasoning traces and final solutions in a training environment, retaining only successful ones for fine-tuning. We experiment with **RFT-Distill**, where a smaller student learns from a stronger teacher, and **RFT-Self**, where a model learns from its own successful trajectories.

4.1 Experimental Setup

Models. We use four student models from the Qwen3 family (Qwen Team, 2025): Qwen3-4B, Qwen3-8B, and Qwen3-14B in thinking mode, and Qwen3-4B-Instruct as a non-thinking variant. For expert distillation, we use Qwen3-32B as the teacher.

Training. For each model, we sample 16 trajectories per task in D3-Gym, yielding 6,780 trajectories per model, retaining only those that pass the silver evaluation scripts. The number of successful trajectories reflects growing capability with model size: Qwen3-4B (1,126), Qwen3-8B (1,306), Qwen3-14B (1,788), and Qwen3-32B (2,153). In **RFT-Distill**, students are fine-tuned on successful trajectories from the Qwen3-32B teacher, and in **RFT-Self**, models are fine-tuned on their own successful trajectories. For Qwen3-4B-Instruct, we apply only RFT-Distill using the final solutions without reasoning traces.

Evaluation. We evaluate on **ScienceAgent-Bench** (Chen et al., 2025), where a model must generate a Python program that processes input data, implements the required analysis, and saves results to the correct output path. We also evaluate on **ScienceAgentBench-Verified**, a manually veri-

Method	ScienceAgentBench				ScienceAgentBench-Verified			
	SR	VER	SR@3	VER@3	SR	VER	SR@3	VER@3
<i>Qwen3-4B-Instruct</i>								
Base	4.2	25.8	5.9	35.3	4.2	25.8	5.9	35.3
RFT-Distill	7.2 (+3.0)	30.4 (+4.6)	8.8 (+2.9)	47.1 (+11.8)	7.2 (+3.0)	30.4 (+4.6)	8.8 (+2.9)	47.1 (+11.8)
<i>Qwen3-4B</i>								
Base	4.9	19.6	6.9	27.5	5.2	20.1	7.3	29.2
RFT-Distill	7.2	23.5	8.8	37.3	7.2	23.2	8.8	36.3
RFT-Self	6.9 (+2.0)	23.2 (+3.6)	11.8 (+4.9)	40.2 (+12.7)	6.9 (+1.7)	23.2 (+3.1)	11.8 (+4.5)	38.2 (+9.0)
<i>Qwen3-8B</i>								
Base	6.5	29.7	11.8	43.1	8.2	31.2	13.8	44.7
RFT-Distill	8.5	31.7	14.7	48.0	8.2	29.1	13.7	43.1
RFT-Self	10.1 (+3.6)	36.9 (+7.2)	12.8 (+1.0)	43.1 (0.0)	10.8 (+2.6)	33.7 (+2.5)	13.7 (-0.1)	41.2 (-3.5)
<i>Qwen3-14B</i>								
Base	9.1	35.3	13.7	49.0	10.3	36.3	15.0	50.0
RFT-Distill	10.5	35.0	13.7	48.1	10.5	33.0	13.7	47.1
RFT-Self	13.1 (+4.0)	37.6 (+2.3)	19.6 (+5.9)	54.9 (+5.9)	14.1 (+3.8)	37.3 (+1.0)	20.6 (+5.6)	54.9 (+4.9)
<i>Qwen3-32B</i>								
Base	14.7	36.3	19.6	50.9	15.0	37.0	20.0	51.0
RFT-Self	18.9 (+4.2)	43.8 (+7.5)	27.4 (+7.8)	58.8 (+7.9)	19.3 (+4.3)	41.8 (+4.8)	28.4 (+8.4)	54.9 (+3.9)

Table 2: Main results on ScienceAgentBench and ScienceAgentBench-Verified (% , $\uparrow$). We report average Success Rate (SR) and Valid Execution Rate (VER) across three independent runs, as well as best-of-three performance (SR@3 and VER@3). For RFT-Self, numbers in parentheses indicate absolute change from the corresponding base model. Best results within each model block are bolded.

fied version introduced here, where we fixed nine unclear instructions, two erroneous gold programs, and one redundant gold execution result (Appendix D). We report **Success Rate (SR)**, whether outputs satisfy human-annotated success criteria, and **Valid Execution Rate (VER)**, whether the program executes without errors and produces output in the correct location. For each model we conduct 3 independent inference runs and report both average and best-of-3 performance. Further training details are in Appendix C.

4.2 Training Results

Training on D3-Gym consistently improves performance across model sizes. As shown in Table 2, training on trajectories from D3-Gym improves both SR and VER across all model sizes, indicating models learn to produce programs that are both executable and scientifically correct. Gains are evident not only in average performance but also in best-of-3 performance (SR@3), reflecting broader capability gains rather than only improved typical-case performance. These gains scale with model size: SR@3 improves by +1.0, +5.9, and +7.8 for the 8B, 14B, and 32B models, respectively.

Models learn effectively from their own trajectories. Across models, RFT-Self generally matches

or outperforms RFT-Distill, with the gap widening at larger scales. For Qwen3-4B and Qwen3-8B, neither approach is consistently better. At 14B, however, RFT-Self becomes consistently superior, improving SR@3 by 5.9 points while RFT-Distill yields no improvement, even degrading VER@3. A plausible explanation is that on-policy trajectories are better aligned with the model’s own distribution (Zhao et al., 2026). To verify this, we score self-generated and 32B teacher-generated responses under each RFT-Self model on 64 randomly sampled prompts, using negative log-likelihood (NLL) and perplexity (PPL); lower values mean the response is more likely under that model’s own distribution. Both Qwen3-8B (NLL: 0.252 vs. 0.652; PPL: 1.29 vs. 1.93) and Qwen3-14B (NLL: 0.261 vs. 0.543; PPL: 1.30 vs. 1.73) assign substantially lower values to self-generated responses, which are preferred on all prompts in both cases. This supports that self-generated trajectories are closer to the model’s learned distribution, enabling more effective learning.

D3-Gym-32B approaches the performance of larger and proprietary models. As shown in Figure 1, training on D3-Gym enables smaller models to match or surpass significantly larger ones. Notably, D3-Gym-32B outperforms both

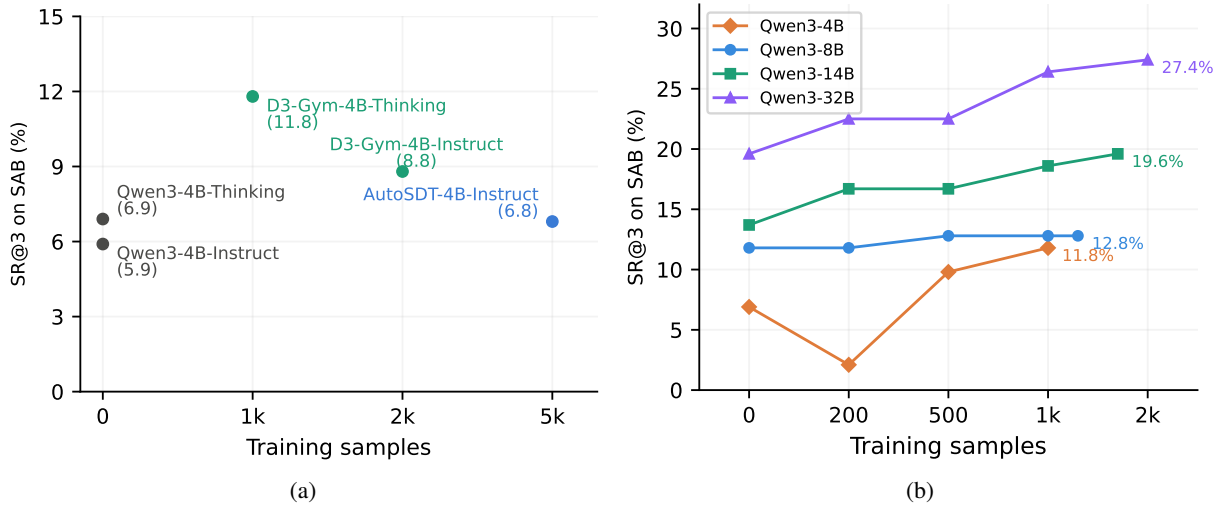


Figure 4: (a) Training on AutoSDT-5K. (b) RFT-Self scaling across models.

Qwen3-80B-A3B and Qwen3-235B-A22B, two larger mixture-of-experts reasoning models, and narrows the gap with the strong proprietary o1-preview (OpenAI, 2024) and Claude Sonnet 4.5 (Anthropic, 2025b). Similarly, D3-Gym-14B sees substantial gains, matching Qwen3-32B despite being less than half its size.

4.3 Analysis

Training on D3-Gym outperforms static SFT data. Figure 4a compares training on D3-Gym data against AutoSDT-5K (Li et al., 2025), a static dataset of 5K instruction–solution pairs that lacks executable environments or verification signals. When used for standard SFT, D3-Gym is more sample-efficient: Qwen3-4B-Instruct trained on D3-Gym solutions achieves roughly 30% relative improvement over its AutoSDT-5K baseline using less than half the training samples, likely due to the rigorous filtering that guarantees task quality in D3-Gym. Beyond sample efficiency, D3-Gym’s executable environment and verification signals enable sampling full reasoning trajectories, unlocking thinking model training; Qwen3-4B improves from 6.9% to 11.8% SR@3 when trained on its own verified reasoning trajectories.

Larger models show more consistent improvement with training scale. Figure 4b shows SR@3 under RFT-Self as training trajectories increase. The 14B and 32B models improve steadily with additional data, whereas the 8B model gains marginally before plateauing and the 4B model initially degrades with few training samples before recovering. This suggests larger models better ex-

tract useful signal from increasing training data, consistent with previous studies (Li et al., 2025; Jain et al., 2025).

Performance improves across diverse task categories. Figure 5a breaks down SR@3 by task category for the Qwen3-32B base model and D3-Gym-32B. Training yields gains across many categories including computational analysis, data and map visualization, geospatial analysis, and statistical analysis. This breadth suggests D3-Gym tasks are sufficiently diverse to expand model capabilities across scientific domains rather than overfitting to a narrow subset of skills.

Failure modes are greatly mitigated, but some persist. We manually analyze errors from both models across six types: *Data schema*, *API & library misuse*, *Data type & shape*, *File path*, *Logical & algorithmic*, and *Visualization*. As shown in Figure 5b, D3-Gym-32B reduces data schema errors by 44.7%, data type/shape errors by 60.4%, and visualization errors by 15.0%, improving execution validity. Logical and algorithmic errors show a slight increase: with more programs passing execution, the model more frequently produces runnable but semantically incorrect code. API & library errors also remain prevalent despite a modest decrease, suggesting a deep limitation in current LMs’ parametric knowledge.

5 Autoresearch with D3-Gym Environments

Beyond training, D3-Gym environments can also support studying automated research with coding

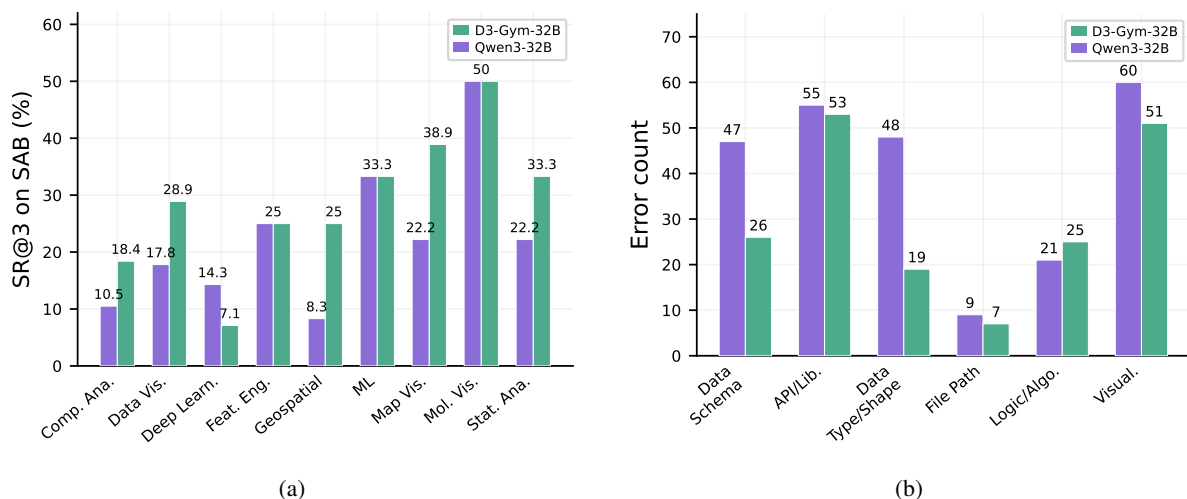


Figure 5: (a) Per-category SR@3 on ScienceAgentBench. (b) Error-type breakdown for Qwen3-32B vs. D3-Gym-32B.

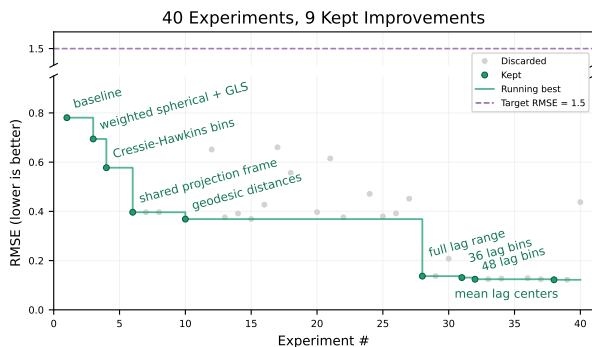
agents: many data-driven discovery tasks can be framed as optimization problems, where an agent repeatedly proposes solutions and improves them against an executable evaluation signal. Unlike the training experiments in §4, where silver evaluation scripts act as binary filters over sampled trajectories, here we expose their raw score and study whether it is reliable enough to guide a frontier coding agent across many iterations.

Setup. We adopt the Autoresearch harness (Karpathy, 2026) with modifications to adapt it to Codex (OpenAI, 2025) and enable persistent memory across sessions, and run Codex with GPT-5.4 for 40 iterations on two D3-Gym tasks. The first is a spatial interpolation problem: given temperature readings from 73 German weather stations, predict the temperature on a regular grid covering the whole country. The prescribed method is universal kriging (Cressie, 1993), essentially Gaussian-process regression with a trend term as its mean function, here a north-south temperature gradient. Solutions are scored by the RMSE of their temperature map against the reference solution’s (lower is better; original pass threshold $\text{RMSE} < 1.5$). The second is more open-ended: train a graph neural network with Chebyshev-polynomial spectral filters (Defferrard et al., 2016) to predict 12 quantum-chemical properties of small molecules, scored by MAE averaged over the properties (threshold $\text{MAE} < 2.0$). Iteration 1 is the agent’s single-shot solution and serves as the baseline. Task specifications and harness details are in Appendix E.

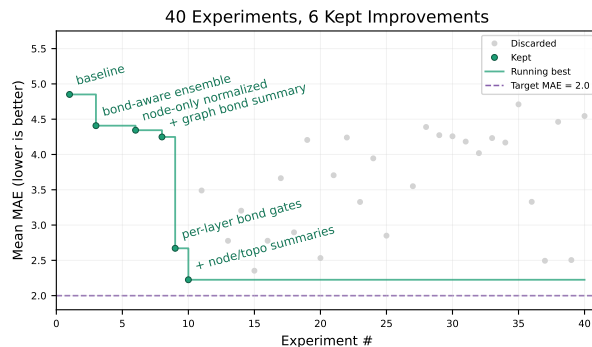
Spatial interpolation task. As shown in Figure 6a, the baseline (RMSE 0.7806) already satisfies the threshold, yet the raw RMSE remains informative and continues to guide improvement. The agent never departs from the prescribed method; every change it keeps refines how the model is fit to the data: a better estimate of how temperature similarity decays with distance, fitting that is less sensitive to outlying stations, consistent geographic coordinates with distances measured along the Earth’s surface, and progressively finer discretization of the station distances (the figure labels each change). The score falls from 0.7806 to 0.1221, a $6.4\times$ reduction, showing that the evaluation scripts synthesized by D3-Gym are reliable enough to serve as a stable optimization objective.

ChebyNet task. Here the agent must make modeling, representation, and training decisions rather than refine a fixed procedure. Its early gains come from neural-network engineering: ensembling, richer inputs (bond information, molecule-level summary features), feature normalization, and gating messages by bond type. MAE falls from 4.8505 to 2.2249, a $2.2\times$ reduction, but then saturates short of the threshold. The cause is a systematic failure: the training set is small, so the reference solution trains a very small model for a single epoch, whereas the agent keeps scaling the model up and training it for hundreds of epochs. This overfitting is never diagnosed or corrected; the agent continues to add architectural complexity even as the score plateaus.

Together, the case studies show that silver evalu-



(a) Spatial Interpolation of German Temperature Data.



(b) Molecular Property Prediction with ChebyNet.

Figure 6: Autoresearch optimization trajectories on two D3-Gym tasks over 40 iterations.

ation scripts give agents reliable signal to improve substantially over their own single-shot solutions, while open-ended tasks surface failure modes such as overfitting and weak self-diagnosis. Automatically constructed verifiable environments can therefore serve not only as training infrastructure but also as testbeds for studying how frontier coding agents search, improve, and fail on realistic scientific workflows.

6 Related Work

Environment creation for coding tasks. Recent work generates executable training environments for coding tasks. In software engineering, SWE-Gym (Pan et al., 2025), RepoST (Xie et al., 2025), R2E-Gym (Jain et al., 2025), and SWE-smith (Yang et al., 2025b) build environments from real GitHub repositories, leveraging commits and unit tests for task synthesis and verification. Beyond software engineering, MLE-Dojo (Qiang et al., 2025), MLGym (Nathani et al., 2025), and MLE-Smith (Qiang et al., 2026) target machine learning tasks where correctness is assessed via standard metrics like F1 and RMSE. Scientific repositories offer neither: codebases rarely include test suites, and correctness is inseparable from the scientific domain, varying with each task’s data modality and analytical goal. Evaluation logic must therefore be constructed from scratch per task.

Environments for data-driven discovery. Most closely related are recent efforts targeting data-driven discovery. Li et al. (2025) introduce a pipeline for collecting scientific coding tasks at scale to construct SFT datasets, but the AutoSDT dataset lacks the execution environments and evaluation scripts needed for trajectory generation or solution validation. We build on their task collection

pipeline and address this limitation by constructing executable, verifiable environments around the collected tasks. Nie et al. (2026) introduce DS-Gym, drawing tasks from existing benchmarks, bioinformatics queries, and Kaggle competitions, with evaluation inherited from these sources. D3-Gym differs in both sourcing and verification: tasks are drawn directly from real scientific repositories across four disciplines, and evaluation scripts are automatically synthesized.

Evaluation script generation. Recent works tackle automated test and evaluation generation for general coding, both to improve test quality (He et al., 2025; Liu et al., 2025) and to enable downstream training (Zeng et al., 2025; Wang et al., 2025; Prasad et al., 2025). There, correctness largely reduces to string or near-string comparison against a reference. Our setting differs fundamentally: evaluation logic for scientific data-driven discovery is tightly coupled to the task’s domain, analytical goal, and data modality, requiring reasoning about which metrics, thresholds, and output artifacts are scientifically appropriate.

7 Conclusion

We introduce D3-Gym, the first automatically constructed dataset of real-world verifiable environments for scientific data-driven discovery, and show its support for both model training and iterative agent refinement. D3-Gym opens a path toward scientific agents that can test, improve, and learn in realistic research settings.

Limitations

We recognize the following limitations and future work directions:

D3-Gym Environment Scale. D3-Gym currently comprises 565 tasks, which is limited compared to training environments in other domains such as software engineering. However, this reflects the difficulty of sourcing high-quality, scientifically meaningful tasks rather than a constraint of our workflow, which can readily collect more tasks provided sufficient repositories are available. Moreover, D3-Gym’s executable and verifiable nature enables sampling thousands of training trajectories, which our results confirm is sufficient to yield substantial improvements across model sizes.

Training method. We focus on RFT for training, which provides dense supervision that is particularly effective when base model performance is weak, as is the case for the open-weight models we train. Our results confirm the effectiveness of this approach in expanding model capabilities across all scales, and we further show that D3-Gym supports training-free agent refinement. While our verifiable environments would also support other training techniques including reinforcement learning, models’ limited baseline capabilities, the complexity of data-driven discovery tasks, and the sparsity of the reward signal make this a challenging direction that we leave for future work.

Evaluation. We evaluate the models trained on D3-Gym on ScienceAgentBench (Chen et al., 2025), as it is the most suitable benchmark that matches D3-Gym’s target capability, namely producing executable programs for real-repository scientific data analysis and judging them against expert-annotated success criteria. We regard evaluation across a wider set of benchmarks, as suitable ones emerge in this space, as an important direction for future work.

Ethical Considerations

We make a concerted effort to ensure D3-Gym tasks are adapted from publicly available scientific repositories with permissive licenses. A complete per-repository license breakdown is in Appendix F.

Author Contributions

HNM led the project, implemented the D3-Gym creation workflow, conducted quality evaluation, model training, and analysis, and wrote the manuscript. YL co-led the project, managed the curation of the validation set for quality evaluation, conducted training and evaluation experi-

ments, contributed to the development of SAB-verified, and assisted with manuscript writing. ZL assisted with initial candidate task collection and co-managed the annotation of the validation set. YY, CT, and TZ contributed to the annotation of the validation set. NKA and AP provided feedback during biweekly project discussions. ZC provided guidance on project direction and ideas, contributed to the development of SAB-verified, explored Autoresearch with D3-Gym, and revised the manuscript. HS advised the project and provided guidance, contributed to core ideas, and revised the manuscript.

Acknowledgments

The authors would like to thank colleagues from the OSU NLP group and Amazon AGI team for constructive feedback. The authors thank Xinming Tu for reaching out and helping to construct ScienceAgentBench-Verified. This research was sponsored in part by NSF OAC 2112606, Cisco, and Ohio Supercomputer Center (Ohio Supercomputer Center, 1987). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

- Anthropic. 2025a. [Claude code](#). Command-line tool for agentic coding.
- Anthropic. 2025b. [Introducing Claude Sonnet 4.5](#).
- Akari Asai, Jacqueline He, Rulin Shao, Weijia Shi, Amanpreet Singh, Joseph Chee Chang, Kyle Lo, Luca Soldaini, Sergey Feldman, Mike D’Arcy, David Wadden, Matt Latzke, Jenna Sparks, Jena D. Hwang, Varsha Kishore, Minyang Tian, Pan Ji, Shengyan Liu, Hao Tong, and 9 others. 2026. [Synthesizing scientific literature with retrieval-augmented language models](#). *Nature*, 650:857–863.
- Frazier N. Baker, Daniel Adu-Ampratwum, Reza Averly, Botao Yu, Huan Sun, and Xia Ning. 2025. [Larc: Towards human-level constrained retrosynthesis planning through an agentic framework](#). *Preprint*, arXiv:2508.11860.
- Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan

- Sun. 2025. [Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery](#). In *The Thirteenth International Conference on Learning Representations*.
- Noel A. C. Cressie. 1993. *Statistics for Spatial Data*, revised edition. Wiley Series in Probability and Statistics. John Wiley & Sons, New York.
- Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems* 29, pages 3844–3852.
- Dayuan Fu, Shenyu Wu, Yunze Wu, Zerui Peng, Yaxing Huang, Jie Sun, Ji Zeng, Mohan Jiang, Lin Zhang, Yukun Li, Jiarui Hu, Liming Liu, Jinlong Hou, and Pengfei Liu. 2026. [davinci-env: Open swe environment synthesis at scale](#). *Preprint*, arXiv:2603.13023.
- Zhongmou He, Yee Man Choi, Kexun Zhang, Jiabao Ji, Junting Zhou, Dejia Xu, Ivan Bercovich, Aidan Zhang, and Lei Li. 2025. [Hardtests: Synthesizing high-quality test cases for llm coding](#). *Preprint*, arXiv:2505.24098.
- Tony Hey, Stewart Tansley, Kristin Tolle, and Jim Gray. 2009. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research.
- Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. 2025. [R2e-gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents](#). In *Second Conference on Language Modeling*.
- Andrej Karpathy. 2026. autoresearch. <https://github.com/karpathy/autoresearch>. GitHub repository, accessed March 31, 2026.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- Yifei Li, Hanane Nour Moussa, Ziru Chen, Shijie Chen, Botao Yu, Mingyi Xue, Benjamin Burns, Tzu-Yao Chiu, Vishal Dey, Zitong Lu, Chen Wei, Qianheng Zhang, Tianyu Zhang, Song Gao, Xuhui Huang, Xia Ning, Nesreen K. Ahmed, Ali Payani, and Huan Sun. 2025. [AutoSDT: Scaling data-driven discovery tasks toward open co-scientists](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 30396–30418, Suzhou, China. Association for Computational Linguistics.
- Yifei Liu, Li Lyna Zhang, Yi Zhu, bingcheng dong, Xudong Zhou, Ning Shang, Fan Yang, Cheng Li, and Mao Yang. 2025. [rstar-coder: Scaling competitive code reasoning with a large-scale verified dataset](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Sanchaita Hazra, Ashish Sabharwal, and Peter Clark. 2024. [Position: Data-driven discovery with large generative models](#). In *Forty-first International Conference on Machine Learning*.
- Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. 2025. [Discoverybench: Towards data-driven discovery with large language models](#). In *The Thirteenth International Conference on Learning Representations*.
- Ludovico Mitchener, Jon M Laurent, Benjamin Tenmann, Siddharth Narayanan, Geemi P Wellawatte, Andrew White, Lorenzo Sani, and Samuel G Rodrigues. 2025. [Bixbench: a comprehensive benchmark for llm-based agents in computational biology](#). *Preprint*, arXiv:2503.00096.
- Siddharth M. Narayanan, James D. Braza, Ryan-Rhys Griffiths, Albert Bou, Geemi Wellawatte, Mayk Caldas Ramos, Ludovico Mitchener, Samuel G. Rodrigues, and Andrew D. White. 2025. [Training a scientific reasoning model for chemistry](#). *Preprint*, arXiv:2506.17238.
- Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Mikhail Plekhanov, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Nicolaus Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. 2025. [MLGym: A new framework and benchmark for advancing AI research agents](#). In *Second Conference on Language Modeling*.
- Fan Nie, Junlin Wang, Harper Hua, Federico Bianchi, Yongchan Kwon, Zhenting Qi, Owen Queen, Shang Zhu, and James Zou. 2026. [Dsgym: A holistic framework for evaluating and training data science agents](#). *Preprint*, arXiv:2601.16344.
- Ohio Supercomputer Center. 1987. [Ohio supercomputer center](#).
- OpenAI. 2024. [Learning to reason with LLMs](#).
- OpenAI. 2025. Introducing codex. <https://openai.com/index/introducing-codex/>.
- Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2025. [Training software engineering agents and verifiers with SWE-gym](#). In *Forty-second International Conference on Machine Learning*.
- Archiki Prasad, Elias Stengel-Eskin, Justin Chen, Zaid Khan, and Mohit Bansal. 2025. [Learning to generate unit tests for automated debugging](#). In *Second Conference on Language Modeling*.

- Rushi Qiang, Yuchen Zhuang, Yinghao Li, Dingu Sagar V K, Rongzhi Zhang, Changhao Li, Ian Shu-Hei Wong, Sherry Yang, Percy Liang, Chao Zhang, and Bo Dai. 2025. [MLE-dojo: Interactive environments for empowering LLM agents in machine learning engineering](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- Rushi Qiang, Yuchen Zhuang, Anikait Singh, Percy Liang, Chao Zhang, Sherry Yang, and Bo Dai. 2026. [MLE-smith: Scaling MLE tasks with automated multi-agent pipeline](#). In *The Fourteenth International Conference on Learning Representations*.
- Qwen Team. 2025. [Qwen3 technical report](#). Preprint, arXiv:2505.09388.
- Swarnadeep Saha, Xian Li, Marjan Ghazvininejad, Jason E Weston, and Tianlu Wang. 2025. [Learning to plan & reason for evaluation with thinking-LLM-as-a-judge](#). In *Forty-second International Conference on Machine Learning*.
- Parshin Shojaei, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. 2025a. [LLM-SR: Scientific equation discovery via programming with large language models](#). In *The Thirteenth International Conference on Learning Representations*.
- Parshin Shojaei, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D Doan, and Chandan K. Reddy. 2025b. [LLM-SRBench: A new benchmark for scientific equation discovery with large language models](#). In *Forty-second International Conference on Machine Learning*.
- Michael D. Skarlinski, Sam Cox, Jon M. Laurent, James D. Braza, Michaela Hinks, Michael J. Hammerling, Manvitha Ponnampati, Samuel G. Rodrigues, and Andrew D. White. 2024. [Language agents achieve superhuman synthesis of scientific knowledge](#). Preprint, arXiv:2409.13740.
- Arthur Spirling. 2023. [Why open-source generative AI models are an ethical way forward for science](#). *Nature*, 616:413.
- Yu Su, Diyi Yang, Shunyu Yao, and Tao Yu. 2024. [Language agents: Foundations, prospects, and risks](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts*, pages 17–24, Miami, Florida, USA. Association for Computational Linguistics.
- Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L. Griffiths. 2024. [Cognitive architectures for language agents](#). *Transactions on Machine Learning Research*. Survey Certification, Featured Certification.
- Xinming Tu, Tianze Wang, Yingzhou Lu, Kexin Huang, Yuanhao Qu, and Sara Mostafavi. 2026. [Benchmarkguard: Who guards the benchmarks? automated auditing of llm agent benchmarks](#). Preprint, arXiv:2604.24955.
- Zihan Wang, Siyao Liu, Yang Sun, Ming Ding, and Hongyan Li. 2025. [CodeContests+: High-quality test case generation for competitive programming](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 5576–5600, Suzhou, China. Association for Computational Linguistics.
- Yiqing Xie, Alex Xie, Divyanshu Sheth, Pengfei Liu, Daniel Fried, and Carolyn Rose. 2025. [RepoST: Scalable repository-level coding environment construction with sandbox testing](#). In *Second Conference on Language Modeling*.
- Dayu Yang, Tianyang Liu, Daoan Zhang, Antoine Simoulin, Xiaoyi Liu, Yuwei Cao, Zhaopu Teng, Xin Qian, Grey Yang, Jiebo Luo, and Julian McAuley. 2025a. [Code to think, think to code: A survey on code-enhanced reasoning and reasoning-driven code intelligence in LLMs](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 2586–2616, Suzhou, China. Association for Computational Linguistics.
- John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. 2025b. [SWE-smith: Scaling data for software engineering agents](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. [Scaling relationship on learning mathematical reasoning with large language models](#). arXiv preprint arXiv:2308.01825.
- Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. 2025. [Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model?](#) In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhui Chen. 2025. [ACECODER: Acing coder RL via automated test-case synthesis](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12023–12040, Vienna, Austria. Association for Computational Linguistics.
- Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. 2026. [Self-distilled reasoner: On-policy self-distillation for large language models](#). arXiv preprint arXiv:2601.18734.
- Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyang Luo. 2024. [Llamafactory: Unified efficient fine-tuning of 100+ language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 400–410.

Yuqi Zhu, Yi Zhong, Jintian Zhang, Ziheng Zhang, Shuofei Qiao, Yujie Luo, Lun Du, Da Zheng, Ningyu Zhang, and Huajun Chen. 2025. [Why do open-source llms struggle with data analysis? a systematic empirical study](#). *Preprint*, arXiv:2506.19794.

Appendix

We provide more details omitted from the main text in the Appendix as follows:

- Appendix A: Details of D3-Gym construction workflow
 - Appendix A.1: Pipeline Statistics
 - Appendix A.2: Prompts
 - Appendix A.3: LLM judge agreement with human
- Appendix B: D3-Gym details
 - Appendix B.1: Example tasks
 - Appendix B.2: Cost breakdown
 - Appendix B.3: Quality validation of D3-Gym verification signal
 - Appendix B.4: Examples of Gold and Silver Evaluation Scripts
 - Appendix B.5: Task difficulty
- Appendix C: Training Details
- Appendix D: ScienceAgentBench-Verified
- Appendix E: Autoresearch Experimental Details
- Appendix F: Repository Licenses
- Appendix G: Disclosure of AI use

A Details of D3-Gym construction workflow

A.1 Pipeline Statistics

The creation workflow of D3-Gym follows the steps provided in [section 2](#). Concretely, we begin by collecting 5,111 candidate tasks using the AutoSDT pipeline (Li et al., 2025). After filtering to retain only tasks whose reference solutions operate exclusively on dependencies present in the original repository and generating dataset previews, 1,586 tasks remain. Of these, 1,263 execute successfully in isolated conda environments and produce output artifacts. Finally, 565 pass the multimodal LLM-as-judge output verification and receive silver evaluation scripts, constituting the final D3-Gym environment — approximately 11% of the initial pool,

reflecting the stringent filtering needed to obtain the highest quality tasks.

A.2 Prompts

In this section we present all the agent and LLM instructions used in our workflow, namely for task filtering and preview creation, output validity judgment, and evaluation script planning and generation.

Task Filtering Agent Prompt

You are a coding agent with access to the local file system. You are given a candidate task consisting of a Python program (program.py) and its associated data files. Your objective is twofold: (1) determine whether this task uses only real data and real library dependencies, with no mock, stub, or simulated logic; and (2) if the task is valid, generate dataset preview files for downstream use.

Phase 1: Task Validation. Carry out the following steps to determine task validity:

1. **Identify Data Files.** Inspect the Python program and extract every file path that it attempts to open, read, load, or process. For each path:

- Verify whether the file exists on the current file system.
- Read a small portion of the file to confirm it contains real, meaningful data (not empty, not placeholder, not all zeros, not constant meaningless values, not randomly generated toy content).

If all referenced data files exist and contain valid data, set `dummy_data = 0`; otherwise set `dummy_data = 1`.

2. **Check for Mock or Simulated Logic.** Analyze the program for any of the following:

- Mock objects or `unittest.mock` usage (e.g., `Mock()`, `MagicMock`, `patch`).
- Custom stub or fake class implementations replacing real libraries.
- Synthetic data generation used as primary input instead of loading from files.
- Hardcoded dummy values pretending to be real data.
- `try-except` blocks catching `ImportError` and defining mock replacements.
- Any workaround for missing dependencies that bypasses the real library.

If the program contains any such logic, set `has_mock = 1`; otherwise set `has_mock = 0`.

3. **Render Verdict.** If `dummy_data == 0` and `has_mock == 0`, set `valid = 1`; otherwise set `valid = 0`. If `valid = 0`, skip Phase 2 and proceed directly to the output.

Phase 2: Dataset Preview Generation (only if `valid = 1`).

For each data file referenced by the program, generate a preview file showing only the raw data schema:

- For CSV or tabular data: the header row plus 3–5 example rows.
- For JSON: a snippet showing the structure with 1–2 entries.
- For text files: the first few lines of raw content.
- For binary or image files: a brief factual description (e.g., “PNG images, 224×224, RGB”).

Each preview must follow the format:

```
[START Preview of <file_path>]
<raw data snippet>
[END Preview of <file_path>]
```

Multimodal Gold-Result Judge Prompt

You are a meticulous evaluator for data-driven discovery workflows. Each task represents an analysis workflow on a dataset—such as statistical analysis, machine learning model training, data visualization, or computational simulation.

Inputs Provided.

1. A *task instruction* describing the analysis to perform.
2. *Dataset preview* files showing the structure and content of the input data.
3. A list of output files created in `pred_results/` and their content (a.k.a gold results).

The gold results were generated by synthesized programs. Complete correctness is *not* guaranteed due to possible errors in code synthesis, data processing, or the execution environment.

Evaluation Criteria. For a task to PASS, *all* of the following must hold:

1. **All requested output files must exist.** Read the task instruction to identify every output file that should be produced. Verify each one appears in the output files listing. If any requested file is missing, the task FAILS.
2. **Each output file must be valid.** For every file, verify that:
 - It is non-empty and contains substantive content (no placeholders, no “TODO”, no all-zeros).
 - It contains no error messages or stack traces.
 - Its content aligns with what the task instruction requested.
 - Its format and values are reasonable for the analysis type.

Use your domain knowledge to assess whether the results are plausible (e.g., metric values within expected ranges, predictions that are sensible given the data).

Evaluation Procedure.

1. **Identify required output files** from the task instruction.
2. **Check file existence** against the list of required outputs.
3. **Validate each file** for correctness, completeness, and validity.
4. **Render a final decision:** PASS only if all files exist and all are valid.

Output Format. Return a JSON object with exactly three fields: `{"valid": true|false, "reason": "<explanation>"}`

Evaluation Script Planner Prompt

You are an expert in scientific data analysis and evaluation methodology. Your task is to analyze a data-driven discovery task and produce a concrete evaluation plan that a coding agent will implement.

Inputs Provided.

- **Task Description:** the natural-language instruction defining the analysis to perform.
- **Dataset Information:** the dataset path, folder structure, and schema previews.
- **Expected Output Files:** the list of files the gold program produces.
- **Reference Result Files:** the actual content of the reference outputs.

Instructions. Produce a structured evaluation plan covering the following:

1. **Task Type.** Classify the task (e.g., classification, regression, clustering, visualization, statistical analysis, simulation).
2. **Evaluation Metrics.** Specify which metric(s) are appropriate and justify why:
 - *Classification:* if the dataset is imbalanced, prefer F1-score, balanced accuracy, AUROC, or AUPRC over accuracy.
 - *Regression:* consider MAE, RMSE, R^2 , or domain-specific error metrics.
 - *Statistical analysis:* consider *p*-values, effect sizes, confidence intervals.

- *Clustering:* consider silhouette score, Davies–Bouldin index, or domain-specific quality measures.
- *Other:* specify domain-appropriate criteria.

3. **Success Thresholds.** Define specific threshold values for each metric, grounded in domain standards and task complexity.
4. **Special Considerations.** Note any domain-specific requirements such as handling of missing data, biological versus statistical significance, cross-validation needs, or output format constraints.
5. **Evaluation Steps.** Provide a clear 3–5 step evaluation procedure that the coding agent should follow.

Output Requirements. The plan must be concise, unambiguous, and specify exactly one evaluation strategy. Avoid presenting multiple alternatives or optional branches—the coding agent will implement the plan directly as written.

Evaluation Script Coder Prompt

You are a Python coding assistant. Your task is to generate an evaluation script for a scientific computing benchmark, following the evaluation plan provided below.

Inputs Provided.

- **Evaluation Plan:** the structured plan produced by the evaluation planner.
- **Task Instruction:** the natural-language description of the analysis task.
- **Dataset Information:** the dataset path, folder structure, and schema previews.
- **Expected Output Files:** the list of files the gold program produces in `pred_results/`.
- **Reference Result Files:** the reference output contents in `gold_results/`.

Structure. The generated script must adhere to the following:

1. **Directory layout (hardcoded paths):**
 - Predicted results: `./pred_results/`
 - Reference results: `./reference_results/`
2. **Function signature:** Define a top-level `eval()` function taking no parameters. It returns a tuple (`result`, `message`) where `result` is a boolean (`True/False`) indicating pass/fail and `message` is a string with details.
3. **Visual evaluation** (for tasks producing plots or images): use an LLM-as-a-judge approach to compare the predicted and reference visual outputs.
4. **Main block:**

```
if __name__ == "__main__":
    ok, msg = eval()
    print(ok, msg)
```
5. **Error handling:** Wrap the body of `eval()` in a `try/except` so that unexpected errors return (`False`, `f"Error: {e}"`) rather than crashing.
6. **File existence checks:** Verify that both predicted and gold files exist before loading. Return (`False`, `"Missing file: ..."`) if any file is absent.

Output Format. Respond with *only* the Python source code for the evaluation script. Do not include any explanation or markdown formatting.

A.3 LLM Judge Agreement with Human

To validate the reliability of the LLM-as-judge used for output verification (§2), one author independently annotated 52 task outputs and we measured

agreement with the LLM judge (GPT-5.2).

Table A.1 reports the results. The LLM judge achieves 92.31% raw agreement with the human annotator (48/52) and a Cohen’s κ of 0.85, corresponding to significant agreement. Treating the human labels as ground truth, the judge attains perfect precision (1.00) and 0.87 recall, with an F1 of 0.93. All four disagreements are false negatives, i.e. cases where the LLM rejected outputs that the human accepted, indicating that the judge is conservative rather than permissive. This bias is desirable in our setting, as the judge may discard a small number of valid tasks but does not allow low-quality outputs into D3-Gym tasks.

Metric	Value
Raw Agreement	92.31%
Cohen’s κ	0.85
Precision	1.00
Recall	0.87
F1	0.93

Table A.1: Agreement between the LLM judge (GPT-5.2) and a human annotator on output verification for 52 task instances. Precision, recall, and F1 treat human labels as ground truth.

B D3-Gym Details

B.1 Example Tasks

In this section we present examples of tasks from D3-Gym belonging to the disciplines of geographic information science and computational chemistry. The examples include the task instruction, dataset previews, and excerpts from the reference solution and evaluation script code.

Example Task: Geographic Information Science

Task Instruction. Perform spatial interpolation of German temperature data using Universal Kriging with a north-south drift model. Use the temperature observation data from `temp_obs.txt` and German border coordinates from `de_borders.txt`. First, estimate the empirical variogram of the temperature measurements and fit a Spherical variogram model to characterize the spatial correlation structure. Then apply Universal Kriging to interpolate temperature values across a regular lat-lon grid covering Germany (47–56.1°N, 5–16.1°E with 0.1° resolution), incorporating a linear north-south drift function based on latitude. Save the fitted variogram plot, the interpolated temperature field and mean drift field as CSV files, temperature field visualizations, a north-south cross-section plot, and the variogram model parameters.

Dataset Preview.

- `temp_obs.txt` — 73 DWD weather stations (id, latitude, longitude, temperature in °C):

#	id,	lat,	lon,	temp
44		52.9336	8.2370	15.70
73		48.6159	13.0506	13.90
91		50.7446	9.3450	17.00
96		52.9437	12.8518	21.90

- `de_borders.txt` — German border as lon/lat coordinate pairs:

```
9.52402 47.5242
9.35000 47.5989
9.18281 47.6707
```

Reference Solution (excerpt).

```
import gstools as gs, numpy as np

data = np.loadtxt("benchmark/.../temp_obs.txt")
ids, lat, lon, temp = data.T

# Empirical variogram + Spherical model fit
bin_center, vario = gs.vario_estimate(
    (lat, lon), temp, latlon=True,
    geo_scale=gs.KM_SCALE, max_dist=900)
model = gs.Spherical(latlon=True, geo_scale=gs.KM_SCALE)
model.fit_variogram(bin_center, vario, nugget=False)

# Universal Kriging with latitude drift
def north_south_drift(lat, lon): return lat
uk = gs.krige.Universal(model=model, cond_pos=(lat, lon),
    cond_val=temp, drift_functions=north_south_drift)

g_lat = np.arange(47, 56.1, 0.1) # 92 points
g_lon = np.arange(5, 16.1, 0.1) # 112 points
uk.set_pos((g_lat, g_lon), mesh_type="structured")
uk(return_var=False, store="temp_field")
uk(only_mean=True, store="mean_field")
```

Evaluation Script (excerpt). The script validates the variogram model type and fitted parameters, computes RMSE and R^2 between the predicted temperature field and the gold standard, and checks physical validity of the interpolated values including the expected latitudinal temperature gradient.

```
# Grid dimensions must be exactly (92, 112)
expected_rows = len(np.arange(47, 56.1, 0.1))
expected_cols = len(np.arange(5, 16.1, 0.1))
assert pred_temp.shape == (expected_rows, expected_cols)

# Variogram: must be Spherical; validate fitted parameters
assert "spherical" in model.text.lower()
assert variance > 0 and nugget >= 0 and nugget < variance
assert 10 < len_scale < 5000 # km (gold ~ 596 km)

# Temperature field RMSE vs gold < 1.5; R^2 vs gold > 0.60
temp_rmse = np.sqrt(np.mean((pred_temp - gold_temp)**2))
ss_res = np.sum((pred_temp - gold_temp)**2)
ss_tot = np.sum((gold_temp - np.mean(gold_temp))**2)
temp_r2 = 1.0 - ss_res / ss_tot
assert temp_rmse < 1.5 and temp_r2 > 0.60

# Physical plausibility: temps in [-10, 40] C
assert np.min(pred_temp) >= -10 and np.max(pred_temp) <= 40

# North-south gradient: |T_south - T_north| > 0.5 C
south = np.mean(pred_mean[:10, :]) # lat ~47-48
north = np.mean(pred_mean[-10:, :]) # lat ~55-56
assert abs(south - north) > 0.5
```

Example Task: Computational Chemistry

Task Instruction. Calculate the Madelung constants for a series of ionic crystal structures using the Ewald summation method. Process the crystal structure files `NaCl.vasp`, `CsCl.vasp`, `ZnO-Hex.vasp`, `ZnO-Cub.vasp`, `TiO2.vasp`, and `CaF2.vasp` to determine the electrostatic potential at the reference atom site in each crystal lattice. The Madelung constant quantifies the electrostatic interactions in ionic crystals by summing contributions from both real space and reciprocal space using the Ewald summation technique with appropriate convergence parameters. Save the computed results to a formatted table showing the crystal name, reference atom symbol, and calculated Madelung constant value for each structure.

Dataset Preview.

- Six VASP POSCAR crystal structure files spanning different crystal symmetries:

`NaCl.vasp` — rock salt (face-centered cubic):

```
Na Cl
1.0000000000000000
0.000000 2.820000 2.820000
2.820000 0.000000 2.820000
2.820000 2.820000 0.000000
Na Cl
```

```

1 1
Cartesian
0.000000 0.000000 0.000000
2.820000 0.000000 0.000000

```

TiO₂. vasp — rutile (tetragonal, 6 atoms):

```

Ti 0
1.0000000000000000
4.593700 0.000000 0.000000
0.000000 4.593700 0.000000
0.000000 0.000000 2.958700
Ti 0
2 4
Cartesian
0.000000 0.000000 0.000000
2.296850 2.296850 1.479350
1.400160 1.400160 0.000000
3.193540 3.193540 0.000000
3.697010 0.896690 1.479350
0.896690 3.697010 1.479350

```

Reference Solution (excerpt).

```

from ase.io import read
import numpy as np

ZZ = 'Na': 1, 'Cl': -1, 'Cs': 1, 'Ca': 2, 'F': -1,
      'Ti': 4, 'Zn': 2, 'O': -2

class EwaldSum:
    def __init__(self, atoms, charges):
        self.cell = atoms.get_cell()
        self.positions = atoms.get_positions()
        self.charges = [charges[s] for s in
            atoms.get_chemical_symbols()]

    def get_madelung(self, eta=None):
        # Optimal eta from cell volume
        volume = abs(np.linalg.det(self.cell))
        eta = (np.pi * len(self.atoms) / volume)**(1/3)
        # Sum real-space + reciprocal-space +
        ↪ self-correction
        potential = self._real_space(eta) +
        ↪ self._recip_space(eta)
        potential -= 2 * eta / np.sqrt(np.pi) *
        ↪ self.charges[0]
        return -potential / self.charges[0]

for crys in ['NaCl', 'CsCl', 'ZnO-Hex', 'ZnO-Cub',
    ↪ 'TiO2', 'CaF2']:
    atoms = read(f"benchmark/.../{crys}.vasp")
    M = EwaldSum(atoms, ZZ).get_madelung()

```

Evaluation Script (excerpt). The script compares computed Madelung constants against established literature reference values for each crystal system (e.g., NaCl = 1.7476, CsCl = 1.7627, TiO₂ = 2.408), checking that all six crystals are present, values are positive, and the relative percentage error stays below 5%.

```

# Literature reference values for each crystal system
REFERENCE =
'NaCl': 'value': 1.7476, 'tolerance': 0.05,
'CsCl': 'value': 1.7627, 'tolerance': 0.05,
'ZnO-Hex': 'value': 1.6413, 'tolerance': 0.10,
'ZnO-Cub': 'value': 1.6381, 'tolerance': 0.10,
'TiO2': 'value': 2.408, 'tolerance': 0.15,
'CaF2': 'value': 2.5194, 'tolerance': 0.10,

# All 6 crystals must be present
for crystal in
    ↪ ['NaCl', 'CsCl', 'ZnO-Hex', 'ZnO-Cub', 'TiO2', 'CaF2']:
    assert crystal in pred_results

# Values must be positive; RPE < 5% vs literature
for crystal, (atom, pred_val) in pred_results.items():
    assert pred_val > 0
    ref = REFERENCE[crystal]['value']
    rpe = abs(pred_val - ref) / ref * 100.0
    assert rpe < 5.0

```

Workflow Stage	Cost (\$)
Candidate Task Collection	1,094.5
Filtering & Dataset Preview Creation	404.5
Code Execution	0.0
Output Validation	46.8
Evaluation Planning & Generation	153.3
Total	1,699.1

Table B.2: Cost breakdown (\$) by workflow stage for constructing D3-Gym.

B.2 Cost Breakdown

Table B.2 breaks down the cost of each stage of the workflow for constructing D3-Gym. On average, the cost of constructing one environment in D3-Gym is around \$3. Most of this cost is incurred by the candidate task collection stage using the AutoSDT pipeline (Li et al., 2025) due to its multi-step search and filtering for scientifically meaningful tasks across thousands of repository files. This cost remains within reasonable bounds and is far lower than that of recent work synthesizing training environments for software engineering tasks (Fu et al., 2026).

B.3 Quality Validation of D3-Gym Verification Signal

B.3.1 LLM Judge Prompt

Evaluation Logic Agreement Judge Prompt

You are a scientific computing evaluation expert. You will be given two evaluation scripts written for the same data analysis task: a *gold* (human-written) script and a *silver* (LLM-generated) script. Your job is to rigorously assess whether the silver script makes the same scientific evaluation decisions as the gold. The gold script serves as the authoritative reference; the silver script is being evaluated for alignment with it.

Focus on the substance of the evaluation logic—the scientific properties being measured, the pass/fail criteria, and the artifacts being compared—not on superficial differences such as code style, variable naming conventions, or choice of library. Your assessment should reflect whether the two scripts would reach the same pass/fail conclusion on a correctly implemented solution.

Score each of the following three aspects *independently* on a 1–5 Likert scale:

- **1 – Completely missing or wrong:** the aspect is absent from the silver script, or the silver script’s implementation is entirely incorrect with respect to the gold.
- **2 – Present but fundamentally different:** the silver script addresses the aspect, but in a way that is scientifically incompatible with the gold (e.g., evaluating a different property altogether).
- **3 – Partially aligned:** the silver script captures the intent of the gold, but contains differences substantial enough to change pass/fail outcomes on some inputs.
- **4 – Well aligned:** the silver script is consistent with the gold, with only minor differences unlikely to affect pass/fail outcomes in practice.
- **5 – Fully aligned or equivalent:** the silver script implements the same evaluation logic as the gold, or a functionally equivalent alternative.

Aspect 1: Evaluation Metric. Determine whether the silver script measures the same scientific property as the gold. Consider the metric definition, its mathematical formulation, and the correctness of the implementation. Two scripts that compute the same quantity using different libraries

or formulations are aligned. However, an incorrect implementation of the intended metric (e.g., computing concordance without accounting for censoring in survival analysis, or using accuracy on a heavily imbalanced classification task where the gold uses F1) constitutes a misalignment and should be scored accordingly.

Aspect 2: Acceptance Criteria. For each pass/fail threshold defined in the gold script, determine whether the silver script applies a corresponding threshold on the same or an equivalent metric. Consider whether the thresholds are calibrated such that a correct solution passing the gold script would also pass the silver, and vice versa. Absent thresholds, dramatically different threshold values, or thresholds applied to the wrong metric all indicate misalignment.

Aspect 3: Target Artifact. Determine whether the silver script reads the correct output files, extracts the correct columns, keys, or fields from those files, and loads ground-truth data from the correct reference source. Misalignment includes reading the wrong file, parsing incorrect columns, or comparing against the wrong ground-truth artifact.

Output Format. Return only a JSON object:

```
{
  "evaluation_metric": {"score": <1-5>, "rationale": "<one sentence>"},
  "acceptance_criteria": {"score": <1-5>, "rationale": "<one sentence>"},
  "target_artifact": {"score": <1-5>, "rationale": "<one sentence>"}
}
```

B.3.2 LLM Judge - Human agreement

To validate the LLM-as-judge used for evaluating silver script quality (§2.3), one author independently scored 20 silver evaluation scripts against their gold counterparts on the same three aspects, i.e. metric choice, threshold & tolerance, and target artifact, using the 1–5 Likert scale in §B.3.1. Table B.3 reports agreement with the LLM judge (Claude Sonnet 4.5).

Across all three dimensions, the judge achieves 85% exact agreement and 98% within-one agreement with the human annotator. Agreement is highest on metric choice (90% exact) and lowest on threshold & tolerance (80% exact), though even in the latter case all scores fall within one point of the human rating.

Aspect	Exact Agr.	Within-1 Agr.
Metric Choice	0.90	0.95
Threshold & Tolerance	0.80	1.00
Target Artifact	0.85	1.00
Overall	0.85	0.98

Table B.3: Agreement between the LLM judge (Claude Sonnet 4.5) and a human annotator on metric choice, threshold & tolerance, and target artifact.

B.4 Examples of Gold and Silver Evaluation Scripts

Example 1

Task. Train a model on the Delaney solubility dataset and produce point predictions (y_{pred}) together with calibrated per-molecule uncertainty estimates (y_{std}).

Key differences. The gold script checks $\text{RMSE} \leq 1.75$ and whether $|\text{error}_i| \leq 2\sigma_i$ for $\geq 70\%$ of molecules. The silver script tightens RMSE to 1.70 and switches to the standard 95% CI convention ($\hat{y} \pm 1.96\sigma$), requiring $\geq 72\%$ coverage. For well-calibrated uncertainties the two coverage definitions are nearly equivalent, but the silver thresholds are marginally tighter on both axes.

Gold evaluation script. Uses a 2σ envelope for uncertainty calibration and an RMSE threshold of 1.75.

```
# --- thresholds ---
RMSE_THRESHOLD = 1.75 # <--
# -> slightly more lenient
COVERAGE_THRESHOLD = 0.70

# --- RMSE ---
rmse = float(np.sqrt(np.mean((y_pred - y_true) ** 2)))

# --- uncertainty calibration: 2-sigma envelope ---
abs_error = np.abs(y_pred - y_true)
coverage_2sigma = float(np.mean( # <--
# -> P(|error| <= 2 * std)
abs_error <= 2.0 * y_std))

if rmse > RMSE_THRESHOLD:
    return False, f"RMSE too high: rmse:.4f"
if coverage_2sigma < COVERAGE_THRESHOLD:
    return False, f"P(|error| <=
# -> 2*sigma)=coverage_2sigma:.3f"
```

Silver evaluation script. Uses the standard 95% confidence-interval convention (1.96σ) and a tighter RMSE threshold of 1.70.

```
# --- thresholds ---
rmse_threshold = 1.70 # <-- 0.05
# -> tighter
coverage_threshold = 72.0 #
# -> (percent scale)

# --- RMSE ---
rmse = np.sqrt(np.mean((y_true - y_pred) ** 2))

# --- uncertainty calibration: 95% CI ---
ci_lower = y_pred - 1.96 * y_std # <--
# -> 1.96*sigma CI
ci_upper = y_pred + 1.96 * y_std # <--
coverage = np.mean( # <--
# -> P(y_true in CI)
(y_true >= ci_lower) & (y_true <= ci_upper)) * 100

success = (rmse <= rmse_threshold
and coverage >= coverage_threshold)
```

Example 2

Task. Train a graph convolutional network to predict polymer crystallization tendency from molecular SMILES, evaluated on a held-out test set.

Key differences. The gold script uses a single $\text{MSE} \leq 350$ gate (equivalent to $\text{RMSE} \approx 18.7$). The silver script replaces this with three complementary metrics: $\text{MAE} \leq 25$, $\text{RMSE} \leq 30$, and $R^2 \geq 0.1$ —all must pass. The added R^2 gate catches degenerate models that predict near the mean: such a model could have $\text{MSE} \approx 300$ and pass the gold script, but would fail the silver one due to low variance explained.

Gold evaluation script. Uses a single MSE gate with a generous threshold of 350.

```
MSE_THRESHOLD = 350.0 # <-- single gate

y_true = merged["value"].values.astype(float)
y_pred = merged["predicted_crystallization_tendency"] \
    .values.astype(float)

mse = float(np.mean((y_true - y_pred) ** 2))
```

```

if mse > MSE_THRESHOLD:                # <-- only check
    return False, f"MSE too high: {mse:.4f}"
return True, f"MSE={mse:.4f}"

```

Silver evaluation script. Replaces the single MSE gate with three complementary metrics: MAE, RMSE, and R^2 .

```

MAE_THRESHOLD = 25.0                    # <-- triple gate
RMSE_THRESHOLD = 30.0
R2_THRESHOLD = 0.1

mae = mean_absolute_error(y_true, y_pred)
rmse = np.sqrt(mean_squared_error(y_true, y_pred))
r2 = r2_score(y_true, y_pred)

all_pass = (mae <= MAE_THRESHOLD        # <-- all three
            and rmse <= RMSE_THRESHOLD   # must pass
            and r2 >= R2_THRESHOLD)

```

B.5 Task Difficulty

Table B.4 details the difficulty study summarized in §3. We report for each model the fraction of the 75 sampled tasks whose generated program executes without errors (execution success) and additionally passes the silver evaluation script (evaluation success). The gap between the two columns shows that producing runnable code is not the main bottleneck, as most programs that execute still fail the task-specific criteria.

C Training Details

Training Data Collection. For each model, we sample 16 trajectories per task on all tasks in D3-Gym. Each trajectory contains the model’s full reasoning trace and final solution, and is executed against the corresponding silver evaluation script in the environment. We retain only trajectories that successfully pass the silver evaluation and use them as training data. This yields 1,126 successful trajectories for Qwen3-4B, 1,306 for Qwen3-8B, 1,788 for Qwen3-14B, and 2,153 for Qwen3-32B. In **RFT-Distill**, student models are fine-tuned on the 2,153 successful trajectories generated by Qwen3-32B. In **RFT-Self**, each model is fine-tuned on its own successful trajectories. For Qwen3-4B-Instruct, we only apply RFT-Distill and use the final solutions without reasoning traces.

Supervised Fine-tuning. We perform fine-tuning using the LlamaFactory library (Zheng et al., 2024). We use the Qwen template and train with LoRA rank 16, LoRA alpha 32, and dropout 0.05. Training is performed in the standard supervised fine-tuning (SFT) stage over the retained successful trajectories.

Training Infrastructure. Training is conducted on NVIDIA H100 GPUs. For smaller models (4B/8B/14B), we use 4 H100 80GB GPUs, while

the 32B model is trained on 8 H100 96GB GPUs. We use a learning rate of $5e-5$, cosine learning rate scheduling, 1 training epoch, warmup ratio 0.05, bf16 precision, gradient checkpointing, and max gradient norm 1.0. We set both the training cutoff length and maximum sequence length to 32,768 tokens. Training uses per-device batch size 1 and gradient accumulation steps 1. We use distributed data parallel training for all runs.

Inference and Evaluation. We use vLLM (Kwon et al., 2023) to serve all models for both trajectory collection and evaluation. Unless otherwise specified, we use temperature 0.2, top_p 0.95, and max_tokens 16k. For each evaluation setting, we run 3 independent inference runs and report the average performance across runs.

D ScienceAgentBench-Verified

In this work, we introduce **ScienceAgentBench-Verified**, a manually revised version of ScienceAgentBench (SAB) that corrects a set of errors and ambiguities in the original benchmark. The issues were flagged using BenchGuard (Tu et al., 2026) and then checked and fixed by the authors. The revisions cover task instructions, dataset/environment files, gold programs, and gold results. In total, the verified update revises 9 task instructions (Tasks 9, 12, 26, 29, 31, 34, 35, 67, and 92) and updates gold programs and/or gold results for 3 tasks (Tasks 21, 32, and 78). These changes fix issues such as mismatched output specifications, incorrect file references, ambiguous column names, and inconsistencies between the written instruction and the implemented evaluator.

Summary of benchmark revisions. The main revisions in SAB-Verified are as follows: (i) instruction fixes for Tasks 9, 12, 26, 29, 31, 34, 35, 67, and 92; and (ii) gold-program and/or gold-result updates for Tasks 21, 32, and 78. Representative example revisions include correcting the requested statistic in Task 9, changing the expected output from SMILES to drug names in Task 12, correcting input file names in Tasks 29, 34, and 35, refining the expected output schema in Task 67, and fixing matrix factor definitions in Task 92.

Impact of the benchmark revision. To validate the effectiveness of our verification, we compare several representative frontier models and coding agents on both the original SAB and SAB-Verified. On the original SAB, the strongest success rate

Model	Evaluation Success	Execution Success
GPT 5.4	38.66%	82.66%
Claude Opus 4.6	36.00%	78.66%
Qwen3 235B A22B	29.33%	61.33%
Qwen3 32B	16.00%	36.00%

Table B.4: Evaluation and execution success of different models on a subset of 75 tasks from D3-Gym, showing the non-triviality of the tasks and the room for improvement.

among the evaluated settings is achieved by **Claude Code (Claude-Sonnet-4.6)** at **44.1 SR**. Replacing SAB with SAB-Verified while keeping the environment unchanged generally leads to small but non-negligible shifts in performance, often around 1–2 solved tasks. For example, Claude-Sonnet-4.6 with self-debug improves from 35.3 SR / 82.4 VER to 38.2 SR / 86.3 VER, Claude Code (Claude-Sonnet-4.6) improves from 44.1 SR / 79.4 VER to 45.1 SR / 80.4 VER, and Codex CLI (GPT-5.2) improves from 43.1 SR / 89.2 VER to 45.1 SR / 87.3 VER. After additionally incorporating the affected gold-program and gold-result fixes, the resulting performance changes are generally marginal, typically within about one solved task. This suggests that the verified benchmark mainly improves evaluation fidelity by reducing noise and ambiguity, while largely preserving the overall difficulty of the benchmark.

E Autoresearch Experimental Details

This section provides the harness configuration and the full task specifications for the two Autoresearch case studies presented in [section 5](#). We select two D3-Gym tasks that naturally fit the iterative optimization setting: a classical geostatistical task with a structured objective, and a more open-ended graph-learning task requiring architecture and training choices.

Harness setup. We adopt Karpathy’s Autoresearch harness ([Karpathy, 2026](#)), with minor changes to support Codex ([OpenAI, 2025](#)) and persistent memory across sessions. The original harness assumes a continuous agent session. Codex sessions are more naturally interactive, so we add a lightweight loop mechanism that allows the experiment to continue across sessions until a fixed iteration budget is reached. We also provide the agent with a persistent `result.tsv` file that records previous attempts, scores, and short descriptions, and instruct the agent to read this file before making further changes. This file serves as the agent’s

memory across iterations.

Each environment contains a `problem.md` file with the task instruction and rules, a `data/` directory with the task inputs, and an `evaluation.py` script adapted from the corresponding silver evaluation script. For the experiments here, we modify the evaluation scripts to return the raw optimization score rather than only a binary pass/fail verdict. We use Codex with GPT-5.4 for 40 iterations per task. The first iteration creates a baseline solution from scratch, and the remaining iterations attempt to improve it using the feedback from the evaluator.

Autoresearch Task 1: Spatial Interpolation of German Temperature Data

- **Task.** Perform Universal Kriging on German DWD station temperatures using a north-south latitude drift. The inputs are `temp_obs.txt` (station id, latitude, longitude, temperature) and `de_borders.txt` (German border coordinates). Fit a Spherical variogram, interpolate a 92×112 Germany-wide grid ($47\text{--}56.1^\circ\text{N}$, $5\text{--}16.1^\circ\text{E}$, 0.1° resolution), and save the variogram plot, temperature field, mean drift field, temperature maps, cross-section plot, and model parameters.
- **Optimization score.** Temperature-field RMSE against the reference output; lower is better. The original binary success threshold is $\text{RMSE} < 1.5$.

Autoresearch Task 2: Molecular Property Prediction with ChebyNet

- **Task.** Train a ChebyNet graph neural network on the Tencent Alchemy PyTorch Geometric dataset. Molecules are represented as graphs with atom/node features and bond/edge structure, and the model should use Chebyshev convolutions with polynomial order 5. Train on the development split, predict 12 molecular properties for each validation molecule, and write `pred_results/pred_chebynet.csv` with `gdb_idx` plus `property_0`–`property_11`.
- **Optimization score.** Mean absolute error averaged across all 12 properties after merging predictions with gold outputs by `gdb_idx`; lower is better. The original binary success threshold is $\text{MAE} < 2.0$.

F Repository Licenses

We ensure that all repositories from which D3-Gym tasks are adapted are under permissive licenses. We give the full breakdown of licenses in [Table F.5](#) in addition to the names of the two repositories that are under custom licenses. There are also 39 repositories that do not provide any license information. We assume that they permit use for research purposes.

License	Repository	Count
MIT		99
GNU (GPL, AGPL, LGPL)		43
None		39
BSD		29
Apache		22
CC		4
ISC		1
Custom		2
	BrainIAC	
	DeepDelta	
Total		239

Table F.5: Distribution of repository licenses across all source repositories.

G Disclosure of AI Use

The authors have made use of AI tools for coding assistance and writing refinement.