SCIENCEBUDDY: RECURSIVE-IN-RECURSIVE SELF-IMPROVEMENT FOR INTERACTIVE SCIENTIFIC AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

ScienceBuddy is an interactive scientific research workspace that brings evidence access, computational analysis, and researcher interaction into a single conversational workflow. At its core is **recursive-in-recursive self-improvement**, which couples harness evolution with model reinforcement learning: the **inner recursion** improves the harness with the model fixed, while the **outer recursion** trains the model under the improved harness. Harness evolution shapes training experience, and model learning creates new opportunities for harness adaptation. Given executable scientific tasks and independent verifiers, interaction feedback guides full-program revisions, while fresh verifier-scored rollouts train the model. We examine the resulting execution effects, training effects, and dependence on update order. Across seven scientific task categories, the reported two-cycle system reaches 71.0% Test accuracy from a 43.3% initial baseline. Crossed training and evaluation harnesses distinguish model learning from test-time program replacement; a separate control favors alternating updates over one-shot sequencing at equal proposal and optimizer-update counts. Reply-availability and editing-scope controls further characterize procedural adaptation. These findings connect the framework to observable improvements in scientific task execution, with task and verifier construction outside the evaluated method.

1 INTRODUCTION

Scientific research proceeds through analysis, inspection, and revision. Language-model agents can assist by retrieving evidence, querying databases, and executing computational workflows (Huang et al., 2025; Wang et al., 2024; Laurent et al., 2024). Recent systems also evolve scientific hypotheses and reuse research experience (Gottweis et al., 2026; Lyu et al., 2026). Researchers then clarify assumptions, question conclusions, and request additional checks. These exchanges reveal how scientific work should be conducted and assessed, but correcting an answer within a conversation does not establish improvement across tasks. Sustained improvement requires changes in working procedures and underlying capabilities.

The execution program matters at both stages. A harness determines which evidence the model sees, how tools are called, and when an answer is submitted (Lee et al., 2026; Chen et al., 2026a). During reinforcement learning, these same decisions shape the rollouts from which the model learns (Feng et al., 2025; Ning et al., 2026). An edit that changes immediate execution may therefore also change subsequent training experience. This motivates our question: **What does harness adaptation contribute to a scientific agent's current performance and subsequent model learning, and how does update order affect the result?**

Prior work establishes foundations for this problem (related works are in Section A). BetterTogether alternates prompt and weight optimization (Soylu et al., 2024); MetaClaw, SIA, and Co-Harness couple harness adaptation with model updates (Xia et al., 2026; Hebbar et al., 2026; Chen et al., 2026b). We investigate full-program adaptation through three comparisons: changing the program around a fixed model, the program used during training, and the update order. Holding one component fixed provides a focused view of changes in the other.

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.

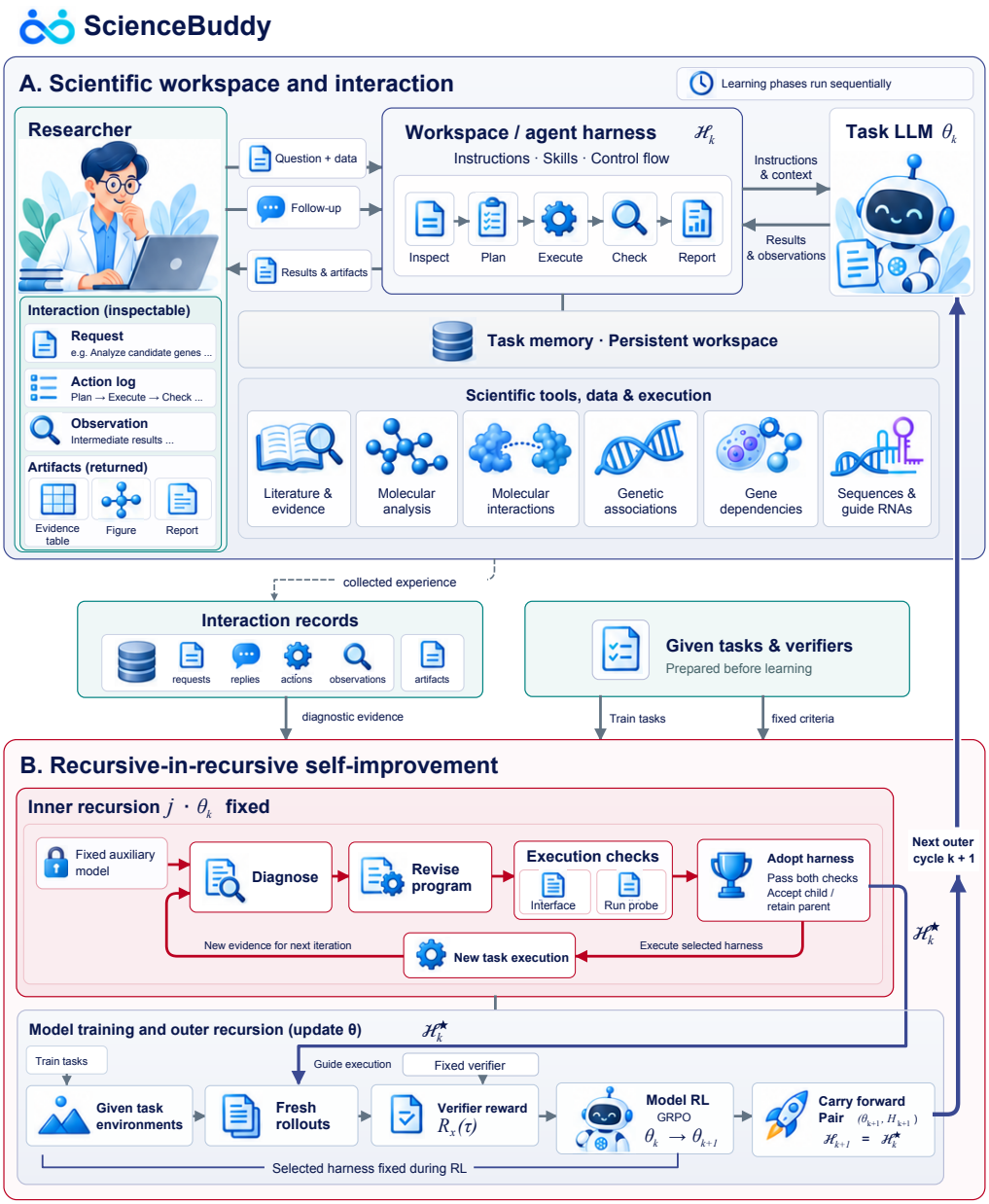


Figure 1: **ScienceBuddy: an interactive scientific workspace with recursive-in-recursive self-improvement.** (A) Researchers submit questions and data, inspect execution traces and artifacts, and refine analyses through follow-up exchanges. (B) User feedback guides harness revision; task verification supervises fresh policy trajectories. Tasks and verifiers are given independently. The evaluated phases run sequentially with interface/preflight adoption.

Setting and scope. Scientific tasks, input assets, execution environments, and independent verifiers are **provided before learning**. Researcher replies guide procedural diagnosis; fixed verifiers evaluate fresh policy rollouts. We do not derive verified RL environments from conversation.

We instantiate the study in **ScienceBuddy**, an **interactive scientific research workspace** combining scientific tools and reference resources with executable analysis, persistent files, and inspectable traces and artifacts. On this foundation, **recursive-in-recursive self-improvement** (R^2SI) couples harness evolution with model reinforcement learning (Figure 1). The **inner recursion** holds the task model fixed while a separate, fixed auxiliary model diagnoses failures and proposes full-program revisions. The **outer recursion** trains on fresh on-policy rollouts under the selected harness. Harness revisions shape training experience, while model updates change the effectiveness of inherited procedures. The updated model then returns to further harness adaptation.

Across seven scientific task categories, the reported two-cycle endpoint improves Test accuracy from 43.3% to 71.0%. Crossed training/evaluation harnesses and serial-versus-alternating controls examine training and scheduling. Reply-availability and editable-scope controls characterize procedural adaptation, including a complete trace-only trajectory under a revised sampling protocol. Researcher cases illustrate the collaboration setting separately from the controlled learning evaluation.

Contributions. Our contributions are fourfold: **(i). A released scientific research workspace.** We develop and release ScienceBuddy, an interactive product that helps researchers carry out scientific tasks by connecting researcher dialogue, executable analysis, inspectable artifacts, and a pluggable harness within a persistent workspace; **(ii). Interaction-grounded tasks and supervision.** We formulate a workflow for deriving executable tasks and evaluation rubrics from collaboration, with capability-aware environment augmentation as a framework extension; **(iii). Recursive-in-recursive self-improvement.** We introduce a paradigm for model–harness co-design that couples evaluated harness evolution with rubric-supervised model reinforcement learning, returning the updated system to researchers for renewed interaction and adaptation. **(iv). Case-study evidence for procedural and model learning.** We examine real researcher interactions, fixed-model harness evolution, and model learning under a fixed harness, relating the proposed framework to improved scientific task execution and broader problem coverage.

2 SCIENCEBUDDY

ScienceBuddy is an interactive scientific research workspace that brings evidence access, computational analysis, and methodological guidance into a single conversational workflow. Researchers can introduce questions together with their data, inspect the resulting analyses, and refine the work through subsequent exchanges. Built on this foundation, ScienceBuddy supports *recursive-in-recursive self-improvement*: an inner process revises and evaluates the agent's harness while keeping the task model fixed (Section 2.3), and an outer process applies continual reinforcement learning to trajectories generated under the evolving harness (Section 2.4). The updated model then returns to further harness adaptation, coupling improvements in working procedures with improvements in the model that executes them (Section 2.5). This section presents the general framework; experiments use its preflight-adoption instance (Figure 1).

2.1 SCIENTIFIC WORKSPACE & AGENT HARNESS

We first describe three system components: scientific tools and execution environments, agent execution and researcher interaction, and modular infrastructure with a pluggable harness. Figure 1 summarizes the scientific workspace and researcher interaction.

Scientific tools and execution environments. ScienceBuddy provides access to a catalog of 224 tools across 22 functional modules, spanning genomics, molecular and cancer biology, pharmacology, bioimaging, literature retrieval, and database queries. The runtime supports Python, R, and Bash execution, combining scientific libraries with data processing, statistical analysis, and visualization. Online database interfaces and a local data lake provide complementary access to biomedical evidence. Researcher-provided documents, tables, sequences, and images enter a persistent workspace that retains inputs, intermediate files, and generated outputs. Interface and environment details appear in Sections C.1 and H.2. The scientific tool catalog and execution utilities are derived from Huang et al. (2025).

Agent execution and researcher interaction. We follow a ReAct-style reasoning–action–observation loop (Yao et al., 2023), alternating reasoning, code or tool execution, and observation. Researchers submit questions, upload supporting data, and provide follow-up instructions through the Chat view. The Trajectory view presents the chronological execution record, an event timeline, and details of selected events. Compute and Results panels provide access to execution activity and generated artifacts. Conversation history and workspace files preserve task context across exchanges, allowing researchers to inspect the agent's work and request revisions. Section H.2 illustrates both interface views.

Modular infrastructure and pluggable harness. ScienceBuddy separates the agent harness from the infrastructure that manages researcher interactions, task execution, and persistent workspaces. A common execution interface specifies the task context supplied to the harness and the responses and execution records returned to the platform. Alternative agentic harnesses can be integrated by implementing this interface, while sharing the same task-management and storage services. Within this architecture, instructions, skills, and selected context-management procedures constitute the editable components of the harness. Recursive improvement revises these components while keeping the surrounding infrastructure fixed, allowing changes in scientific problem-solving procedures to be evaluated under consistent execution conditions (Section 2.3).

2.2 Interaction Formulation and Learning Signals

Interaction formulation. Let x denote a research request and its inputs, π_θ the task model, H the harness, and $h_t = (x, a_0, o_1, \ldots, a_{t-1}, o_t)$ the history, with $h_0 = (x)$. An action a_t is executable code, a tool call, or a researcher-facing response. The observation $o_{t+1} = (e_{t+1}, u_{t+1})$ records environment output or execution status e_{t+1} and an optional researcher reply u_{t+1}, with $u_{t+1} = \perp$ when absent. The harness constructs model context $C_H(h_t)$ from history, memory, skills, and tool descriptions. Allowing for a scheduled deterministic action $d_H(h_t)$, such as input inspection, the joint policy and trajectory are

$$\mu_{\theta,H}(a \mid h_t) = \begin{cases} \delta_{d_H(h_t)}(a), & \text{if a harness action is scheduled,} \\ \pi_\theta(a \mid C_H(h_t)), & \text{otherwise,} \end{cases} \qquad (1)$$
$$a_t \sim \mu_{\theta,H}(\cdot \mid h_t), \qquad \tau = (x, a_0, o_1, \ldots, a_{T-1}, o_T).$$

Here δ denotes a point mass and T counts execution steps. A researcher-facing response may follow several tool steps; a tool observation alone does not constitute a researcher turn or user feedback.

From collaboration to tasks and rubrics. The following construction and SFT routes describe the general framework; experiments use given tasks and verifiers. The collaboration record supplies two complementary artifacts: a self-contained task and its evaluation rubric. Related turns are consolidated around a scientific objective, with independently solvable objectives separated. The task instruction preserves the final requirements and inputs without importing the historical answer. Unlike earlier task-only packaging followed by expert annotation, the current workflow also derives the rubric from the full collaboration trajectory:

$$\mathcal{C}(x) = \mathrm{ConstructRubric}\big(\tau_x^{\mathrm{collab}}; I_x, A_x\big), \qquad (2)$$

where τ_x^{collab} is the source collaboration, I_x the reconstructed instruction, and A_x the required assets. Criteria cover task scope, methodological requirements, evidence, and expected artifacts. Conflicting requirements are resolved before scoring; historical answers and researcher approval are not automatically treated as scientific ground truth.

Harbor tasks for post-training. The task package combines the instruction, input assets, execution environment $\mathcal{E}_x$, and rubric:

$$\mathcal{P}_x = \big(I_x, A_x, \mathcal{E}_x, \mathcal{C}(x)\big). \qquad (3)$$

Instructions, configuration, assets, and rubric-based tests are organized as Harbor tasks (har). The same tasks support two post-training routes: SFT retains rubric-qualified generated trajectories through rejection sampling, while RL collects fresh on-policy rollouts and uses rubric scores as rewards. Input and runtime checks establish executability; rubric-based checks and a fixed judge assess scientific requirements. The rubric remains fixed within each post-training stage.

2.3 Inner Recursion: Feedback-guided Harness Improvement

At inner step j of outer cycle k, the active harness $H_{k,j}$ is the *parent*, and its proposed revision $\widetilde{H}_{k,j+1}$ is a *candidate*. An accepted candidate becomes the *child* $H_{k,j+1}$. Otherwise, the parent remains active.

Feedback-guided diagnosis. Within outer cycle k, the task-model parameters θ_k remain fixed. We use DeepSeek-V4.1-Flash as a separate, fixed auxiliary model for trajectory diagnosis and harness editing. It reviews public trajectories and interpreted user replies, identifies unmet requirements, and cites the relevant actions and observations. Following evidence-based trajectory diagnosis (Barke et al., 2026), it maps these findings to a candidate procedural edit. Private references and verifier outcomes are excluded from its prompts. Task-specific answers and newly supplied facts remain local to the task.

Harness revision. Let $E_{k,j}$ contain the selected public trajectories, interpreted user feedback, and edit history for harness $H_{k,j}$. The auxiliary model proposes a bounded update,

$$\widetilde{H}_{k,j+1} = U(H_{k,j}, E_{k,j}; \theta_k). \tag{4}$$

A proposal can revise skills, instructions, context construction, input inspection, and execution control within the full harness program. Prompt/skill-only editing is evaluated as a restricted control. Interface checks enforce compatibility with the execution system. Tools, execution infrastructure, rubrics, and evaluators remain fixed. This is a procedural update; neither the task model nor the auxiliary model receives gradient updates.

Evaluation and recursive refinement. In the general score-gated variant, parent and candidate are evaluated on identical development tasks and execution conditions using frozen task rubrics. Let $\bar{S}_k(H)$ be the mean normalized rubric score and $\mathrm{Valid}(H)$ indicate compliance with the edit constraints. Write $\Delta_{k,j} = \bar{S}_k(\widetilde{H}_{k,j+1}) - \bar{S}_k(H_{k,j})$. Its acceptance rule is

$$H_{k,j+1} = \begin{cases} \widetilde{H}_{k,j+1}, & \mathrm{Valid}(\widetilde{H}_{k,j+1}) \ \wedge \ \Delta_{k,j} > 0, \\ H_{k,j}, & \text{otherwise.} \end{cases} \tag{5}$$

In this variant, evaluation includes previously successful tasks to account for regressions (Ma et al., 2026), and ties retain the parent. Rejected edits and score changes remain in the optimizer's history. The selected harness then executes new training tasks, whose trajectories supply evidence for the next revision. Iteration continues until the proposal budget or outer collection boundary is reached. Development tasks are separate from policy-training tasks and the final held-out test set. **Experimental instantiation.** In Equation (5), experiments replace the score gate with interface/preflight checks. The last adopted harness carries forward; Test scores never guide adoption.

2.4 Outer Recursion: Continual Model Reinforcement Learning

Environment augmentation under an evolving harness. As the harness evolves, previously challenging tasks may become routine, reducing their value for further model training. The general framework can calibrate environment difficulty through pilot execution with the current task model and selected harness. Following environment evolution (Fan et al., 2026), researcher-derived tasks can be augmented by varying scientific inputs and analysis conditions or extending dependencies between computational steps. Validated environments then supply fresh RL rollouts. Experiments keep the given environments fixed ($\mathcal{T}_k = \mathcal{T}$).

Task-adaptive rubric rewards. For each task x, a fixed rubric composer derives task-specific criteria from the source collaboration trajectory and its reconstructed objective, inputs, and required outputs (Section 2.2), following task-adaptive rubric construction (Ding, 2026). The resulting rubric $\mathcal{C}(x)$ combines task-specific correctness checks with relevant evidence and artifact requirements. Each criterion has a nonnegative importance weight $w_c(x)$, assigned before rollout evaluation, and a satisfaction score $v_c(x, \tau) \in [0, 1]$. We use executable checks where available and a fixed judge for criteria requiring scientific interpretation (Yu et al., 2026). Following rubric-based reward aggregation (Gunjal et al., 2025), the trajectory reward is

$$R_x(\tau) = \frac{\sum_{c \in \mathcal{C}(x)} w_c(x)\, v_c(x, \tau)}{\sum_{c \in \mathcal{C}(x)} w_c(x)}, \qquad \sum_{c \in \mathcal{C}(x)} w_c(x) > 0. \tag{6}$$

Experiments use the fixed verifier as the sole unit-weight criterion, so Equation (6) reduces to $R_x(\tau) = V_x(\tau)$; no judge is used. Rubrics remain fixed during optimization. The terminal reward supplies a trajectory-level advantage shared across generated tokens. We use GRPO (Shao et al., 2024); its objective and implementation details are given in Section B.2.

Model updates and renewed harness adaptation. At outer cycle k, we maximize the expected trajectory reward under the selected harness:

$$\max_{\theta} J_k(\theta), \qquad J_k(\theta) = \mathbb{E}_{x\sim q_k}\mathbb{E}_{\tau\sim\pi_{\theta,H_k^\star}(\cdot|x)}\big[R_x(\tau)\big]. \tag{7}$$

Here q_k samples the given 548 Train tasks in our experiments, and $\pi_{\theta,H_k^\star}$ is the trajectory distribution induced by the task model under the fixed harness $H_k^\star$. The GRPO update yields θ_{k+1}. Because harness effectiveness depends on its interaction with the task model (Lee et al., 2026), we re-evaluate the selected harness under the updated model before deploying (θ_{k+1}, H_{k+1}), with $H_{k+1} = H_k^\star$. Researcher interactions with this pair provide evidence for the next inner-adaptation phase and outer update cycle (Section 2.5).

2.5 Coordinating Recursive-in-Recursive Improvement

Nested update schedule. ScienceBuddy serves researchers with model θ_k and harness H_k over a fixed collection interval. The resulting interactions and feedback initiate a background update cycle, asynchronous with the online service: harness improvement proceeds with θ_k fixed, followed by model RL under the selected harness. After re-evaluation, the updated model–harness pair is deployed to support increasingly demanding research tasks. Subsequent researcher interactions provide the evidence for the next cycle (Algorithm 1). All harness versions and task environments are retained for subsequent evolution. Inherited harnesses are re-evaluated under the updated task model before deployment or reuse.

Algorithm 1 General recursive-in-recursive workflow with asynchronous online service

Require: Initial (θ_0, H_0), collection interval Δ, training environments $\mathcal{T}$, development tasks, inner budgets J_k, RL budgets, and outer count K

Initialize evidence buffer $\mathcal{B}$ and edit history $\mathcal{L}$; deploy (θ_0, H_0)
for $k = 0, \ldots, K-1$ **do**
  $E_k \leftarrow \mathrm{Collect}_\Delta(\theta_k, H_k)$; $\mathcal{B} \leftarrow \mathcal{B} \cup E_k$
  Background updates; the online service continues with (θ_k, H_k).
  $H_{k,0} \leftarrow H_k$; evaluate $\bar{S}_k(H_{k,0})$; $j \leftarrow 0$
  while $j < J_k$ and the inner execution budget remains **do**
    Append fresh task evidence under $(\theta_k, H_{k,j})$ to $\mathcal{B}$
    $E_{k,j} \leftarrow \mathrm{Read}(\mathcal{B}, \mathcal{L}; H_{k,j})$
    $\widetilde{H}_{k,j+1} \leftarrow U(H_{k,j}, E_{k,j}; \theta_k)$ ▷ Section 2.3
    Validate and, if valid, evaluate $\widetilde{H}_{k,j+1}$ under paired development conditions
    Select $H_{k,j+1}$ by Equation (5); record the decision in $\mathcal{L}$
    $j \leftarrow j + 1$
  end while
  $H_k^\star \leftarrow H_{k,j}$; $\mathcal{T}_k \leftarrow \mathrm{Augment}(\mathcal{T}; \theta_k, H_k^\star)$
  $\theta_{k+1} \leftarrow \mathrm{RLUpdate}(\theta_k; H_k^\star, \mathcal{T}_k, R)$ ▷ Section 2.4
  Use fresh batches $\mathcal{D}_{k,t}$, Equations (6) and (9); retire batches after optimization.
  $H_{k+1} \leftarrow H_k^\star$; re-evaluate (θ_{k+1}, H_{k+1})
  Retain all harness and environment versions; $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}_k$
  Deploy (θ_{k+1}, H_{k+1})
end for
return (θ_K, H_K)
Experiments use preflight adoption and given environments without augmentation.

3 Experiments

We first measure the joint endpoint, then distinguish execution-side, training-side, and update-order effects. Reply-availability and editing-scope comparisons examine how the programs are obtained.

Table 1: Main Test results from 210 attempts per condition on 70 tasks. Outcomes are percentages; Δ is change from H0 in percentage points, computed before rounding. Overall weights categories by task count; category average weights them equally. Bold/underline mark highest/second-highest distinct values per row, separately for outcomes and Δ; ties share marks.

Task / measure	H0	Harness-only		RL-only		**ScienceBuddy (R2SI)**	
		Acc.	Δ	Acc.	Δ	Acc.	Δ
Accuracy by scientific task							
Database QA	40.0	55.6	+15.6	51.1	+11.1	**84.4**	**+44.4**
Genetics QA	23.3	51.7	+28.3	36.7	+13.3	**78.3**	**+55.0**
Troubleshooting	**76.7**	66.7	-10.0	66.7	-10.0	**76.7**	**+0.0**
Error Detection	73.3	46.7	-26.7	66.7	-6.7	**80.0**	**+6.7**
Step Ordering	46.7	20.0	-26.7	53.3	+6.7	**66.7**	**+20.0**
Physical/Quantum Chem.	**40.7**	25.9	-14.8	33.3	-7.4	**40.7**	**+0.0**
Other Chemistry	38.9	38.9	+0.0	**44.4**	**+5.6**	**44.4**	**+5.6**
Overall accuracy and submission							
Overall	43.3	47.6	+4.3	47.6	+4.3	**71.0**	**+27.6**
Category average	48.5	43.6	-4.9	50.3	+1.8	**67.3**	**+18.8**
Submission rate	78.1	93.3	+15.2	91.4	+13.3	**96.2**	**+18.1**

All learning results use provided scientific tasks and verifiers; researcher interactions in Section H are separate qualitative context.

3.1 Experimental Setting

Tasks, models, and budgets. The main experiment uses 548 Train and 70 Test tasks across seven categories, selected from LAB-Bench, Biomni, BioProBench, and QCBench (Laurent et al., 2024; Huang et al., 2025; Liu et al., 2025; Xie et al., 2025). Task packages include inputs, execution environments, and fixed answer checkers (Table 5). Qwen3.5-4B is the solver; DeepSeek-V4.1-Flash is the fixed auxiliary model. H0 is the initial system. Harness-only uses 30 proposals with the initial solver; RL-only uses 40 updates under H0; the joint system uses 14 proposals and 30 updates. These component references have different learning budgets. The separate update-order comparison matches proposal and optimizer-update counts.

Evaluation protocols. Main results, stage checkpoints, and mechanism controls use repeated evaluations on the 70-task Test split at temperature zero (Table 1). Feedback is available only during harness adaptation, never during RL or Test. Accuracy counts correct first submissions, including failed or unsubmitted attempts in the denominator. Overall weights category accuracies by Test task counts; the category average gives each category equal weight. Test outcomes never guide updates or adoption.

Comparison scope. Evaluation repeats characterize fixed model and program checkpoints, not repeated training or search. Learning stages, diagnostic controls, and the update-order experiment are identified separately below and in Section D.

3.2 Overall Scientific Task Performance

The reported joint endpoint reaches **71.0% Overall accuracy**, compared with **43.3% for H0**, 47.6% for Harness-only and RL-only (Table 1). The task-weighted endpoint gain over H0 is 27.62 percentage points. Database QA and Genetics QA show the largest category gains. Paired outcomes change from incorrect to correct in 32.4% of attempts and in the reverse direction in 4.8% (Figure 2(c)). The endpoint establishes the performance of this learning path; the following controls examine its components. Figure 2(a,b) traces the two-cycle learning path, ending at the 71.0% result in Table 1. The first harness endpoint is below H0, whereas later stages attain higher accuracy. Preflight adoption therefore does not imply monotonic scientific improvement.

3.3 Execution Effects with Model Weights Fixed

Adaptation can change performance before RL. The Harness-only diagnostic keeps the solver fixed throughout search. Test accuracy is 43.3% at H0 and 57.14% after six proposals, the first measured positive checkpoint with evaluations every three proposals (Figure 8). Two recorded GTRD and ClinVar cases connect Train feedback to reusable data-reading procedures. On different Test

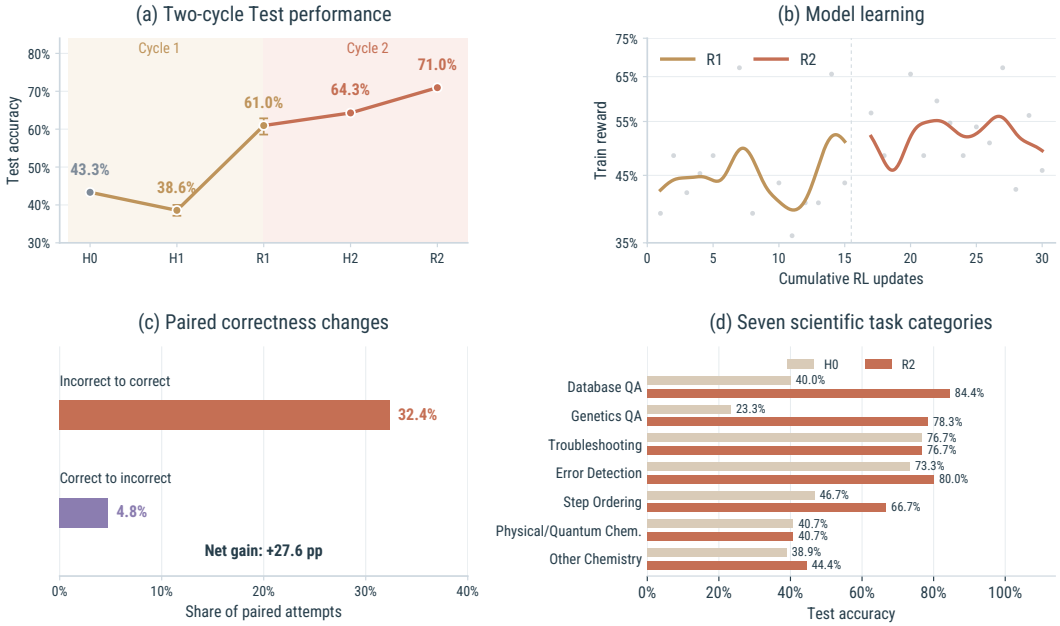


Figure 2: **Stage dynamics and endpoint performance.** (a) Repeated stage evaluations at temperature zero; whiskers show recorded evaluation ranges where available. (b) Train reward on a logarithmic axis with phase-wise smoothing ($\sigma = 1$). (c) Paired changes in correctness across evaluation repeats. (d) Reported category scores in Table 1. The Test panels use the same 70-task split.

Table 2: **Execution, training, and update-order controls.** H0/R2 follow Table 1. Accuracy is in percent; Δ is variant minus reference before rounding. All comparisons use repeated evaluations on 70 Test tasks at temperature zero. Training branches share M1 and fifteen RL updates; schedules use fourteen proposals and thirty updates, with potentially different sampling compute.

Comparison (reference → variant)	Reference	Variant	Δ
A. Execution: model weights fixed			
M0: initial H0 → generic repair	43.3	40.00	−3.33
M2: generic repair → adapted H2	46.67	71.0	+24.29
B. Training: evaluation harness fixed			
Test H1: train under H1 → train under H2	60.48	66.67	+6.19
Test H2: train under H1 → train under H2	61.43	71.0	+9.52
C. Update order: proposal and update counts fixed			
One-shot sequential → alternating two cycles	54.29	71.0	+16.67

questions, the complete evolved harness changes both outcomes from consistently incorrect to consistently correct, without another correction or model update (Section F.1). These selected cases demonstrate reuse by the complete harness, without isolating individual edits.

Submission recovery does not explain every program effect. Table 2-A summarizes execution effects. A generic answering/error-handling repair increases M0's submission rate from 78.1% to 91.43%, while accuracy decreases from 43.3% to 40.00%. At fixed M2, the evolved H2 reaches 71.0%, versus 46.67% for the repair (Table 6); the difference is 24.29 points. Thus, **the evolved program contributes beyond this generic repair**. This comparison does not identify which learned edit accounts for the difference.

3.4 Training Effects under a Common Test Harness

To distinguish training experience from test-time program replacement, two branches start from M1 and the same task cursor and take fifteen RL updates under inherited H1 or adapted H2. Both learned solvers are then **evaluated under each Test harness** (Table 2-B). Under common H2, the H2-trained solver scores 71.0% versus 61.43%, a 9.52-point difference. Under common H1, the difference is 6.19 points. The point estimates favor training under H2; independent learning repeats remain necessary to assess the consistency of this effect. Equal optimizer-update counts also allow

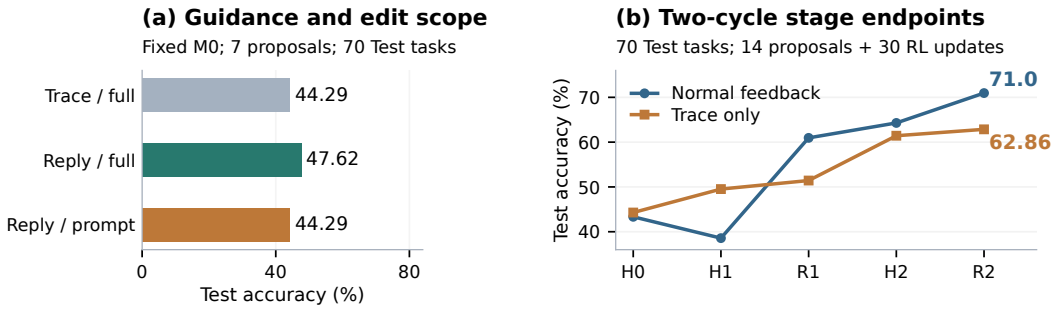


Figure 3: **Feedback evidence at two scales.** (a) Fixed-model reply-availability and editing-scope comparisons. (b) Two-cycle stage endpoints. Both use repeated evaluations on 70 Test tasks at temperature zero. Trace-only R2 revises the refill cap, so sampling compute is not matched. Full counts and protocols are in Section D.

different numbers of sampled attempts. A distinct execution comparison fixes the already-trained M1 and replaces H1 with H2, increasing accuracy from 60.95% to 64.29% (Table 10). H2 also helps M0, by 10.00 points versus 3.33 at M1. Thus, a useful program revision need not specifically repair a mismatch caused by the model update. The crossed comparisons keep this execution-side observation separate from the training-side question.

3.5 Alternating versus One-Shot Sequential Updates

Table 2-C compares schedules with fourteen harness proposals and thirty RL updates. Alternating H7→RL15→H7→RL15 reaches **71.0%**, compared with **54.29%** for H14→RL30, a 16.67-point advantage in the recorded comparison. The sequential branch preserves the reference's two fifteen-update RL blocks, optimizer reset, and sampling recipe; only the location of the second harness phase changes relative to RL. This compares update order at matched proposal and optimizer-update counts, not matched sampled tokens or runtime. The schedule control uses the same 548-task Train and 70-task Test splits. Repeated evaluations characterize fixed checkpoints; consistency across independent learning runs remains untested (Section D.3). The result supports this alternating path without establishing that alternation is universally necessary.

3.6 Reply Guidance and Editable Program Scope

Figure 3(a) compares guidance and editing scope. With M0 fixed, the user-reply condition reaches 47.62%, versus 44.29% for trace-only guidance. The difference is 3.33 points. Restricting edits to prompts and skills also yields 44.29%; the full-program difference is 3.33 points, with submission increasing from 80.00% to 94.29%. These are directional observations from the recorded searches, not conclusive advantages across repeated adaptation runs. The comparisons isolate reply availability and editable scope within the reported settings; full protocol details and outcome counts are in Section D.

Trace-only guidance across two complete cycles. A completed trace-only trajectory reaches **62.86%**, versus **71.0%** with normal feedback on the same 70-task Test panel at temperature zero (Figure 3(b)), an 8.10-point gap. Repeated evaluations provide supporting endpoint evidence; the sampling difference prevents isolating the causal effect of replies. Section D.2 reports stage endpoints and protocol details.

4 Conclusion

ScienceBuddy provides an interactive scientific workspace in which researcher collaboration can inform both working procedures and model learning. Its recursive-in-recursive framework connects evaluated harness refinement with rubric-supervised model updates, returning the updated system to further scientific interaction. The case studies illustrate the complementary contributions of these components: researcher requests and follow-up requirements define scientific tasks and assessment criteria; harness revision improves first-response accuracy with the task model fixed; and model learning expands problem coverage under the initial harness. These findings support the framework's procedural and model-learning mechanisms and provide a basis for studying their coordination across continued researcher collaboration.

AI USE STATEMENT

LLMs are components of the studied algorithm: they simulate user replies, interpret public interaction evidence, propose harness programs, and execute tasks as the solver. Generative AI tools also assisted manuscript drafting and restructuring, comparison of existing literature, interpretation and presentation of recorded results, and figure-authoring code. Reported outcomes come from recorded experimental artifacts, including the completed trace-only run. The authors are responsible for checking AI-assisted content and for the final claims and artifacts.

REPRODUCIBILITY STATEMENT

The method specifies the editable program interface, information boundaries, adoption rule, and solver-training objective. The appendix records task composition, sampling and update budgets, run lineage, stage measurements, and interaction cases. Main, stage, schedule-control, and full trace-only protocols are distinguished explicitly. Figure measurements retain source paths and hashes in the accompanying local evidence records; the manuscript source also includes a trace-only endpoint and protocol summary.

REFERENCES

Harbor: Task Structure. https://www.harborframework.com/docs/tasks. Accessed September 9, 2026.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. *arXiv preprint arXiv:2507.19457*, 2025. doi: 10.48550/arXiv.2507.19457. URL https://arxiv.org/abs/2507.19457.

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. AgentRx: Diagnosing AI Agent Failures from Execution Trajectories. *arXiv preprint arXiv:2602.02475*, 2026. URL https://arxiv.org/abs/2602.02475.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, Yuan Xie, Heng Qu, Kun Shao, and Jian Luan. HarnessX: A Composable, Adaptive, and Evolvable Agent Harness Foundry. *arXiv preprint arXiv:2606.14249*, 2026a. URL https://arxiv.org/abs/2606.14249.

Zhengyu Chen, Teng Xiao, Huaisheng Zhu, Yige Yuan, Luan Zhang, and Jingang Wang. Co-Harness: Co-Evolving Harnesses and Model Weights for LLM Agents. *arXiv preprint arXiv:2607.22688*, 2026b. URL https://arxiv.org/abs/2607.22688.

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. ScienceAgentBench: Toward Rigorous Assessment of Language Agents for Data-Driven Scientific Discovery. *arXiv preprint arXiv:2410.05080*, 2024. doi: 10.48550/arXiv.2410.05080. URL https://arxiv.org/abs/2410.05080.

Liang Ding. AdaRubric: Task-Adaptive Rubrics for Reliable LLM Agent Evaluation and Reward Learning. *arXiv preprint arXiv:2603.21362*, 2026. URL https://arxiv.org/abs/2603.21362.

Tianyu Fan and Chao Huang. HELIX: Model-Harness Co-evolution for Recursive Self-Improvement. *arXiv preprint arXiv:2608.13951*, 2026. URL https://arxiv.org/abs/2608.13951.

Zhiyuan Fan, Tinghao Yu, Yuanjun Cai, Jiang Zhou, Jiangtao Guan, Jincheng Liu, Yun Yang, Dingxin Hu, Zhuo Han, Xing Wu, Feng Zhang, and Lilin Wang. Environment Evolution for Terminal Agents. *arXiv preprint arXiv:2609.04128*, 2026. URL https://arxiv.org/abs/2609.04128.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. ReTool: Reinforcement Learning for Strategic Tool Use in LLMs. *arXiv preprint arXiv:2504.11536*, 2025. URL https://arxiv.org/abs/2504.11536.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, et al. Accelerating Scientific Discovery with Co-Scientist. *Nature*, 2026. doi: 10.1038/s41586-026-10644-y. URL https://doi.org/10.1038/s41586-026-10644-y.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as Rewards: Reinforcement Learning Beyond Verifiable Domains. *arXiv preprint arXiv:2507.17746*, 2025. URL https://arxiv.org/abs/2507.17746.

Eric Han, Jun Chen, Karthik Abinav Sankararaman, Xiaoliang Peng, Tengyu Xu, Eryk Helenowski, Kaiyan Peng, Mrinal Kumar, Sinong Wang, Han Fang, and Arya Talebzadeh. Reinforcement Learning from User Feedback. *arXiv preprint arXiv:2505.14946*, 2025. doi: 10.48550/arXiv.2505.14946. URL https://arxiv.org/abs/2505.14946.

Prannay Hebbar, Yogendra Manawat, Samuel Verboomen, Alesia Ivanova, Selvam Palanimalai, Kunal Bhatia, and Vignesh Baskaran. SIA: Self Improving AI with Harness & Weight Updates. *arXiv preprint arXiv:2605.27276*, 2026. URL https://arxiv.org/abs/2605.27276.

Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf Roohani, Ryan Li, Lin Qiu, Junze Zhang, Yin Di, et al. Biomni: A General-Purpose Biomedical AI Agent. *bioRxiv*, 2025. doi: 10.1101/2025.05.30.656746. URL https://www.biorxiv.org/content/10.1101/2025.05.30.656746v1.

Borui Kang, Jinrui Gu, Junhan Lv, Wenbin Li, Lei Wang, and Yang Gao. Harness Continual Learning: Continual Adaptation Beyond Model Parameters. *arXiv preprint arXiv:2608.19013*, 2026. URL https://arxiv.org/abs/2608.19013.

Jon M. Laurent, Joseph D. Janizek, Michael Ruzo, Michaela M. Hinks, Michael J. Hammerling, Siddharth Narayanan, Manvitha Ponnapati, Andrew D. White, and Samuel G. Rodriques. LAB-Bench: Measuring Capabilities of Language Models for Biology Research. *arXiv preprint arXiv:2407.10362*, 2024. doi: 10.48550/arXiv.2407.10362. URL https://arxiv.org/abs/2407.10362.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-End Optimization of Model Harnesses. *arXiv preprint arXiv:2603.28052*, 2026. doi: 10.48550/arXiv.2603.28052. URL https://arxiv.org/abs/2603.28052.

Yuyang Liu, Liuzhenghao Lv, Xiancheng Zhang, Li Yuan, and Yonghong Tian. BioProBench: Comprehensive Dataset and Benchmark in Biological Protocol Understanding and Reasoning. *arXiv preprint arXiv:2505.07889*, 2025. URL https://arxiv.org/abs/2505.07889.

Yougang Lyu, Xi Zhang, Xinhao Yi, Yuyue Zhao, Shuyu Guo, Wenxiang Hu, Jan Piotrowski, Jakub Kaliski, Jacopo Urbani, Zaiqiao Meng, Lun Zhou, and Xiaohui Yan. EvoScientist: Towards Multi-Agent Evolving AI Scientists for End-to-End Scientific Discovery. *arXiv preprint arXiv:2603.08127*, 2026. URL https://arxiv.org/abs/2603.08127.

Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. SkillGen: Verified Inference-Time Agent Skill Synthesis. *arXiv preprint arXiv:2605.10999*, 2026. URL https://arxiv.org/abs/2605.10999.

Ludovico Mitchener, Jon M Laurent, Alex Andonian, Benjamin Tenmann, Siddharth Narayanan, Geemi P Wellawatte, Andrew White, Lorenzo Sani, and Samuel G Rodriques. BixBench: a Comprehensive Benchmark for LLM-based Agents in Computational Biology. *arXiv preprint arXiv:2503.00096*, 2025. doi: 10.48550/arXiv.2503.00096. URL https://arxiv.org/abs/2503.00096.

Xuying Ning, Dongqi Fu, Tianxin Wei, Hanqing Zeng, Yuanchen Bei, Bingxuan Li, Zihao Li, Qifan Wang, Xiang Shen, Yifan Wu, Jiayi Liu, Hong Li, Yinglong Xia, Xiangjun Fan, Hanghang Tong, and Jingrui He. EvoHarness-RL: Learning Self-Evolving Runtime Harness for Long-Horizon LLM Agents. *arXiv preprint arXiv:2608.05446*, 2026. URL https://arxiv.org/abs/2608.05446.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017. doi: 10.48550/arXiv.1707.06347. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Woong Shin, Craig A. Bridges, Marshall T. McDonnell, and Rafael Ferreira da Silva. Fantastic Scientific Agents and How to Build Them: AgentBuild for Rietveld Refinement. *arXiv preprint arXiv:2606.12834*, 2026. URL https://arxiv.org/abs/2606.12834.

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. doi: 10.48550/arXiv.2303.11366. URL https://arxiv.org/abs/2303.11366.

Dilara Soylu, Christopher Potts, and Omar Khattab. Fine-Tuning and Prompt Optimization: Two Great Steps that Work Better Together. *arXiv preprint arXiv:2407.10930*, 2024. URL https://arxiv.org/abs/2407.10930.

Dhruv Srikanth, Bingchen Zhao, Dixing Xu, Yuxiang Wu, and Zhengyao Jiang. Recursive Self-Improvement of AI Research Agents. *arXiv preprint arXiv:2609.26457*, 2026. URL https://arxiv.org/abs/2609.26457.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable Code Actions Elicit Better LLM Agents. *arXiv preprint arXiv:2402.01030*, 2024. doi: 10.48550/arXiv.2402.01030. URL https://arxiv.org/abs/2402.01030.

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. OpenClaw-RL: Train Any Agent Simply by Talking. *arXiv preprint arXiv:2603.10165*, 2026a. doi: 10.48550/arXiv.2603.10165. URL https://arxiv.org/abs/2603.10165.

Yinjie Wang, Tianbao Xie, Ke Shen, Mengdi Wang, and Ling Yang. RLAnything: Forge Environment, Policy, and Reward Model in Completely Dynamic RL System. *arXiv preprint arXiv:2602.02488*, 2026b. doi: 10.48550/arXiv.2602.02488. URL https://arxiv.org/abs/2602.02488.

Peng Xia, Jianwen Chen, Xinyu Yang, Haoqin Tu, Jiaqi Liu, Kaiwen Xiong, Siwei Han, Shi Qiu, Haonian Ji, Yuyin Zhou, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. MetaClaw: Just Talk – An Agent That Meta-Learns and Evolves in the Wild. *arXiv preprint arXiv:2603.17187*, 2026. URL https://arxiv.org/abs/2603.17187.

Yang Xiao, Yusong Sun, Haoyi Wu, Wenyang Hui, Wen Da, Zhaokai Luo, Mu Chuan, Yao Hu, Wenjie Li, and Chengyue Jiang. PILOT in the Loop: Live Self-Improvement for Long-Horizon Agents. *arXiv preprint arXiv:2608.26530*, 2026. doi: 10.48550/arXiv.2608.26530. URL https://arxiv.org/abs/2608.26530.

Jiaqing Xie, Weida Wang, Ben Gao, Zhuo Yang, Haiyuan Wan, Shufei Zhang, Tianfan Fu, and Yuqiang Li. QCBench: Evaluating Large Language Models on Domain-Specific Quantitative Chemistry. *arXiv preprint arXiv:2508.01670*, 2025. URL https://arxiv.org/abs/2508.01670.

Ran Xu, Yuchen Zhuang, Yishan Zhong, Yue Yu, Zifeng Wang, Xiangru Tang, Hang Wu, May D. Wang, Peifeng Ruan, Donghan Yang, Tao Wang, Guanghua Xiao, Xin Liu, Carl Yang, Yang Xie,

and Wenqi Shi. MedAgentGym: A Scalable Agentic Training Environment for Code-Centric Reasoning in Biomedical Data Science. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2506.04405.

Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. CoEvolve: Training LLM Agents via Agent-Data Mutual Evolution. *arXiv preprint arXiv:2604.15840*, 2026. URL https://arxiv.org/abs/2604.15840.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Ya-Qi Yu, Hao Wang, Fangyu Hong, Xiangyang Qu, Gaojie Wu, Qiaoyu Luo, Nuo Xu, Huixin Wang, Wuheng Xu, Yongxin Liao, Zihao Chen, Haonan Li, Ziming Li, Dezhi Peng, Minghui Liao, Jihao Wu, Haoyu Ren, and Dandan Tu. Reinforcement Learning with Robust Rubric Rewards. *arXiv preprint arXiv:2605.30244*, 2026. URL https://arxiv.org/abs/2605.30244.

Haobo Zhang, Kelong Mao, Sulong Xu, Simiu Gu, and Zhicheng Dou. Learning from Online User Feedback for Shopping Agents. *arXiv preprint arXiv:2608.11604*, 2026a. doi: 10.48550/arXiv.2608.11604. URL https://arxiv.org/abs/2608.11604.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents. *arXiv preprint arXiv:2505.22954*, 2025a. doi: 10.48550/arXiv.2505.22954. URL https://arxiv.org/abs/2505.22954.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026b. doi: 10.48550/arXiv.2603.19461. URL https://arxiv.org/abs/2603.19461.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. *arXiv preprint arXiv:2510.04618*, 2025b. doi: 10.48550/arXiv.2510.04618. URL https://arxiv.org/abs/2510.04618.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-Adapting Language Models. *arXiv preprint arXiv:2506.10943*, 2025. doi: 10.48550/arXiv.2506.10943. URL https://arxiv.org/abs/2506.10943.

A Related Work

Scientific agents. Co-Scientist evolves hypotheses; EvoScientist accumulates research strategies (Gottweis et al., 2026; Lyu et al., 2026). Biomni supplies scientific resources; ScienceAgentBench and BixBench evaluate executable analyses; MedAgentGym provides biomedical training environments (Huang et al., 2025; Chen et al., 2024; Mitchener et al., 2025; Xu et al., 2026). AgentBuild uses scientist-authored rubrics and curricula (Shin et al., 2026). These systems motivate our scientific workspace; scientific use alone does not distinguish the learning algorithm.

Recursive self-improvement. DGM evolves an agent archive, Hyperagents makes the modification procedure editable, and AIDE^2 recursively improves an AI research agent (Zhang et al., 2025a; 2026b; Srikanth et al., 2026). ScienceBuddy studies a nested dependency between harness adaptation and task-model learning. Its auxiliary improver remains fixed: improved task performance does not imply a stronger improvement mechanism.

Persistent experience in agents. Reflexion retains verbal lessons, GEPA searches over prompts using trajectory reflection, and ACE maintains contextual playbooks (Shinn et al., 2023; Agrawal et al., 2025; Zhang et al., 2025b). Meta-Harness and HarnessX optimize executable harnesses (Lee et al., 2026; Chen et al., 2026a). HCL studies retention under harness updates; PILOT learns reusable procedures during live execution (Kang et al., 2026; Xiao et al., 2026). ScienceBuddy studies how procedural adaptation generates experience for a second, model-level recursion.

Co-evolving programs, models, and data. BetterTogether alternates prompts and weights; SEAL generates adaptation data and update directives (Soylu et al., 2024; Zweiger et al., 2025). MetaClaw combines skill evolution with policy optimization; SIA selects harness/weight updates and training algorithms (Xia et al., 2026; Hebbar et al., 2026). **Co-Harness is especially close:** it alternates validated harness patches with supervised learning from filtered trajectories (Chen et al., 2026b). We use reply-guided full-program revisions, execution-based adoption, and fresh verifier-scored GRPO. EvoHarness-RL learns external-harness access/update policies (Ning et al., 2026); CoEvolve adapts task data, and RLAnything adapts environments and rewards (Yang et al., 2026; Wang et al., 2026b). Our tasks and verifiers stay given. HELIX constructs verified learning records without a reported subsequent model-update cycle (Fan & Huang, 2026). Table 14 compares designs, without implying a performance ranking.

Learning from interaction. RLUF models implicit user reactions and LOFA learns from shopping conversations and purchase outcomes (Han et al., 2025; Zhang et al., 2026a). OpenClaw-RL extracts evaluative and directive signals from the states following agent actions, including user replies (Wang et al., 2026a). AgentRx diagnoses failures from execution evidence (Barke et al., 2026); Rubrics as Rewards studies structured evaluative feedback for RL (Gunjal et al., 2025). User feedback guides our procedural revision, while task verification supervises policy trajectories generated under the active harness. The trace-only and editable-scope controls examine reply availability and the program interface.

B Configured Learning Schedule

The reported run uses preflight adoption. Each of its two harness phases attempts seven proposals and carries the last adopted program forward. Interface checks and execution preflight assess compatibility and executability; they do not require a correct scientific answer. Test results are diagnostic only: they do not determine program adoption or provide RL rewards. The first phase is preserved across a resumed run. Completed-result receipts and program hashes establish the model and harness inherited by the next phase.

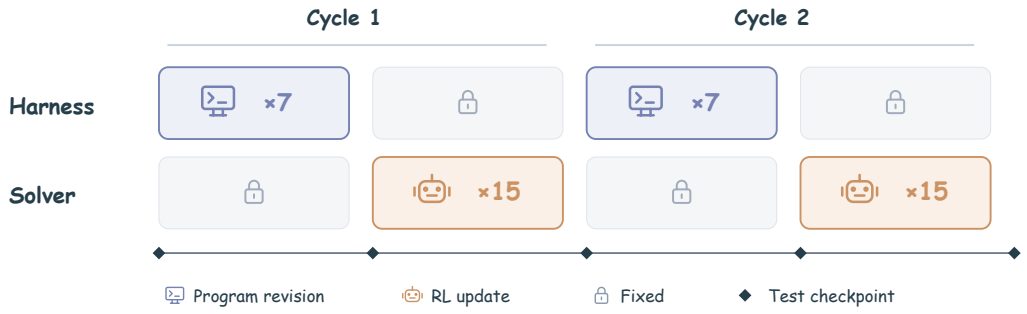


Figure 4: **Reported two-cycle schedule.** Colored cells denote seven harness proposals or fifteen solver updates; locks denote the fixed component. Diamonds denote Test boundaries. Both components carry into the next cycle. The plot depicts the reported two-cycle prefix, not a stopping-time comparison.

Table 3: Learning and evaluation configuration. Model names identify the configured artifacts or API labels.

Component	Setting
Task solver	Qwen3.5-4B
Fixed auxiliary model	DeepSeek-V4.1-Flash
Cycles	2 reported Harness/RL cycles
Harness phase	7 proposals; 16 Train interactions per proposal; 1 candidate
Feedback	At most 1 simulated user reply per Train interaction
Adoption	Interface checks and execution preflight
RL phase	15 updates; 8 task groups $\times$ 8 attempts per update
Baseline schedules	Harness-only: 30 proposals; RL-only: 40 updates
Main Test evaluation	70 tasks; repeated evaluations; temperature 0
RL optimizer	GRPO; learning rate 10^{-6}; 1 epoch per batch; no KL penalty
Task sampling	Temperature 1; top-p 1; thinking disabled
Stage and control evaluation	Temperature 0; repeated evaluations on 70 tasks
Episode limits	9 model calls; 9 tool calls; 4,096 generated tokens per call
Context and observations	24,576 context tokens; 2,048 per tool response; 8,192 tool-history tokens
Tool timeout	60 seconds per Python execution
Backend	SkyRL with FSDP and vLLM
Model handoff	Exported weights and data cursor; optimizer state reset

B.1 Information Available to Each Role

The simulated user and feedback interpreter are separate calls to the configured auxiliary model. The user responds to the public trajectory and may accept an incorrect answer or question a correct one. The interpreter classifies the reply as acceptance, correction, or ambiguity and cites public evidence. The host encodes these classes as $+1$, -1, and 0 for diagnostic records; these values are not model-training rewards. A valid user reply remains available even if its structured interpretation fails. Private references and per-task verifier outcomes are excluded from the user, interpreter, and improver prompts.

Table 4: Learning roles and information boundaries. The auxiliary roles use fixed model weights with separate prompts.

Role	Available evidence	Output or update
Solver	Task, allowed assets, harness context, observed tools and user replies	Model completions and task submissions
Simulated user	Public question, solver response, tools, conversation	Accept, continue, or uncertain with a natural reply
Interpreter	Actual reply and public execution evidence	Feedback intent, supporting observations, uncertainty
Improver	Public Train evidence, parent program, proposal history, adoption-check results	Candidate harness program
Verifier	Submitted output and private task reference	Independent task score
Policy learner	Fresh rollout inputs, generated tokens, log probabilities, verifier reward	Solver parameter update

The improver receives the parent program, public resource inventory, current execution evidence, and historical candidate records. Its interface supports reading allowed evidence fields, proposing a complete program, or declining to propose a change. The current protocol allows at most three evidence reads, two repair attempts, and six auxiliary calls per candidate. Pending trace evidence is cleared after adoption, while aggregate proposal history can persist across model phases. Hypotheses and predicted effects remain model-generated explanations, not causal annotations supplied by a human expert.

B.2 Policy Optimization

For each solver-generated token, let $\rho_{i,t}(\theta)$ be the ratio of its probability under the current policy to that under the collection policy, using its recorded context. The group-relative advantage for attempt i is

$$A_i = \frac{r_i - \bar{r}}{\sigma_r + \epsilon}, \tag{8}$$

where r_i is its task reward and $\bar{r}$ and σ_r are the mean and standard deviation within its task group. The PPO-style clipped surrogate (Schulman et al., 2017) is

$$\mathcal{L}_k(\theta) = \mathbb{E}_{i,t}\left[\min\left(\rho_{i,t}(\theta)A_i, \text{clip}\big(\rho_{i,t}(\theta), 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}}\big)A_i\right)\right]. \tag{9}$$

The pinned SkyRL backend supplies the clipping parameters and masked token reduction. Both the KL-loss term and the KL-reward penalty are disabled in the current recipe. There is one optimizer epoch per rollout batch, with learning rate 10^{-6}. The model receives new rollout groups for subsequent updates.

Each model call becomes a training row containing its original input tokens, completion tokens, and behavior log probabilities. Calls from the same episode share a trajectory identity. The terminal reward is recorded once, at the last completion; the step-wise training adapter broadcasts the corresponding trajectory advantage across its generated-call rows. Prompt and observation tokens are masked out. An episode with no generated model tokens supplies a failure record but no policy-gradient row.

C Scientific Tasks and Evaluation

C.1 Task Composition and Replayable Environments

The latest main comparison in Table 1 uses models retrained on the seven-category, 548-task Train split in Table 5, with 70 held-out tasks used for Test evaluation. Both harness adaptation and solver learning use this Train split. The per-category counts below describe this dataset.

Table 5: Task categories and Train/Test counts used for the retrained models in Table 1. Harness adaptation and solver learning use the 548-task Train split.

Task	Train	Test
Database QA	120	15
Genetics QA	140	20
Troubleshooting	88	10
Error Detection	40	5
Step Ordering	40	5
Physical/Quantum Chem.	78	9
Other Chemistry	42	6
Total	548	70

The general learning loop requires a subset of interactions that can be replayed independently. A task package records the reconstructed instruction, permitted input assets, initial environment, output contract, and frozen verifier. Task-specific facts supplied by a user can enter the reconstructed instruction when they are legitimate inputs; historical solutions and subsequent evaluator outputs cannot be copied into the solver's initial context. Verifier availability is an assumption of this package interface. The current experiments use benchmark references and deterministic checks, rather than claiming automatic verifier construction from arbitrary natural-language feedback.

C.2 Verification and Evaluation Protocol

Database and genetic-evidence tasks use reference-answer checks. Protocol tasks use answer checks or exact step-order comparison, and quantitative chemistry uses a fixed numerical tolerance or interval. Evaluation records distinguish malformed submissions, exhausted budgets, tool errors, and candidate-program errors. The main results use the seven-category split described above. Task counts establish the dataset inventory; they do not by themselves certify that every task is solvable or that scientific source material is independent across splits.

The host enforces fixed inference budgets. Accuracy counts correct first submissions, including failed and unsubmitted attempts in the denominator. Submission rate counts attempts with a submitted answer. Overall weights category accuracies by Test task counts, while Average gives each of the seven evaluated categories equal weight.

D Additional Experimental Results

D.1 Feedback and Program Controls

Table 6: Independent fixed-solver diagnostic experiments. Test accuracy is in percent; Δ is the second condition minus the first in percentage points. Bold/underline mark the higher/lower accuracy within each row.

Comparison (first $\rightarrow$ second)	First	Second	Δ
Fixed M0: H0 $\rightarrow$ generic repair	**43.3**	<u>40.00</u>	-3.33
Fixed M2: generic repair $\rightarrow$ H2	<u>46.67</u>	**71.0**	+24.29

Supplement protocol and generic repair. Table 2-A reports the controls on the common 70-task Test panel. The normal-feedback H1 and reported model endpoints are reused historical controls, not newly repeated learning runs. With the base model, the generic repair scores 40.00% at 91.43% submission. With M2, it scores 46.67% at 90.95% submission, versus 71.0% and 96.2% for H2. The inherited-H1 RL branch submits on 87.14% of analyzed attempts. The repair changes only generic answering and error-handling behavior within the existing execution budgets; it adds no scientific task-specific rules or model updates. Both repair evaluations use the same frozen program, verified by its hash. Source receipts and program changes are retained.

Feedback and editable scope. Table 7 reports three variants: full-program adaptation with user replies or trace-only evidence, and prompt/skill-only adaptation with user replies. Each starts from M0/H0 and completes seven proposals after 112 Train interactions before evaluation through repeated evaluations on 70 Test tasks at temperature zero. The archived identity records match the starting model, harness, code, dataset, runtime, verifier, and improver configuration. Guidance mode is an intervention within this block; total auxiliary-token costs need not match. No model RL occurs in these comparisons.

The user-reply/full-program condition reaches 47.62% versus 44.29% for trace-only evidence and 44.29% for prompt/skill-only editing. Their differences are both 3.33 points. These comparisons concern reply availability and editing scope; independent searches are needed to assess whether the observed differences persist.

Table 7: Reply-availability and editable-surface comparisons at fixed M0. Each uses seven proposals and 210 Test attempts. Values are percentages; bold identifies the full-program, reply-guided condition.

Editable surface	Guidance	Acc. ↑	Submit ↑
Full program	Trace only	44.29	95.71
Prompt/skill only	User reply	44.29	80.00
Full program	User reply	**47.62**	**94.29**

D.2 Complete Trace-Only Learning Trajectory

Each stage endpoint is evaluated repeatedly at temperature zero. It removes simulated user replies from both harness phases and retains fourteen proposals and thirty solver updates. Table 9 reports the five stage endpoints; R2 finishes at 62.86%.

Endpoint comparison and interpretation. The comparison uses the same 70-task Test panel. Normal feedback answers 149/210 correctly, matching the 71.0% R2 reference in Table 1; trace-only answers 132/210 attempts correctly (62.86%), giving an 8.10-point difference computed before rounding. This is one search/training path per condition with repeated endpoint evaluations. Trace-only improves from 44.29% at H0 to 62.86% at R2, but the comparison does not establish that replies alone cause the endpoint gap.

Sampling revision and completed-stage reuse. Completed H1, R1, and H2 stages are reused from the declared ancestry. Following an interruption, R2 restarts from the saved M1 checkpoint and inherited H2, with a fresh optimizer and RL cursor 15; the ten unsaved updates of the failed R2 attempt are not resumed. The completed R2 retains mixed-reward groups under the same frozen sampling policy and draws fresh Train batches until eight such groups are collected, removing the previous four-input-batch cap. Optimizer-update counts are unchanged, but sampling exposure differs from the original normal-feedback protocol. No Test score triggers the retry or selects the endpoint. Failed-attempt costs are recorded separately from the completed path.

Endpoint execution. At the endpoint, 15/210 attempts terminate through output truncation without submitting an answer. The completed R2 rerun including evaluation takes 6,898 seconds (about 1 hour 55 minutes).

D.3 Model–Harness Compatibility

The new M0/H2 measurement completes Table 10. H2 improves over H1 by 10.00 points at M0 and 3.33 points at M1. It therefore benefits both tested models; the larger gain at M0 does not support the stronger claim that H2 specifically repairs a mismatch introduced by the M1 update. Training the solver under a program and evaluating a fixed solver with that program answer different questions. Table 11 reports the completed post-training crossed evaluations on the same 70-task Test panel.

Table 8: Completed two-cycle feedback comparison using repeated evaluations on 70 Test tasks at temperature zero. Normal feedback uses the aggregate R2 reference in Table 1. Both paths use 14 proposals and 30 RL updates. Trace-only R2 revises the mixed-group refill cap, so training sampling compute is not matched.

Condition	Correct / attempts	Test accuracy (%)
Normal feedback (Table 1)	149 / 210	**71.0**
Trace-only (revised R2 refill)	132 / 210	62.86

Table 9: Trace-only stage endpoints on the 70-task diagnostic panel. Each row has 210 attempts.

Stage	Correct / attempts	Test accuracy (%)
H0	93 / 210	44.29
H1	104 / 210	49.52
R1	108 / 210	51.43
H2	129 / 210	61.43
R2 (final)	132 / 210	62.86

Table 10: Crossed model–harness Test accuracy (%). Bold/underline mark the higher/lower accuracy in each model row. Δ is H2 minus H1 in percentage points. Only M0/H2 is newly evaluated in this suite; the remaining cells reuse the reported trajectory.

Solver	H1	H2	Δ
M0	<u>38.57</u>	**48.57**	+10.00
M1	<u>60.95</u>	**64.29**	+3.33

Table 11: Crossed evaluation of second-cycle solvers on the seven-category Test panel through repeated evaluations at temperature zero. Bold/underline compare trained solvers under each fixed Test harness. These training-side diagnostics use the same Test split; bold/underline denote point estimates, not significance.

Second-cycle training harness	Test with H1	Test with H2
Inherited H1	<u>60.48</u>	<u>61.43</u>
Adapted H2	**66.67**	**71.0**

Both second-cycle branches start from M1 and run 15 optimizer updates. Under a common H2, the solver trained with H2 scores 71.0% versus 61.43% for the solver trained with H1, a 9.52-point difference. Holding H1 fixed for evaluation gives a 6.19-point difference. The shared evaluation harness separates these learned-solver comparisons from test-time program replacement. Independent learning repeats remain necessary to assess consistency.

Alternating versus one-shot sequential updates. The sequential control starts from the reference run's M0/H0, performs all 14 harness proposals under M0, then freezes the last adopted harness for 30 RL updates. Both schedules retain the reference's two 15-update RL blocks, optimizer reset, sampling and group-filter recipes, and carried task cursor. They use equal proposal and optimizer-update counts; actual sampled tokens and runtime can differ. This diagnostic uses the same 548-task Train split. Both endpoints are scored on the same 70-task Test panel through repeated evaluations at temperature zero.

The sequential endpoint answers 114/210 attempts correctly (54.29%); the alternating reference is the 71.0% R2 result in Table 1. The resulting 16.67-point difference supports the alternating schedule in this comparison. It compares one completed sequential learning run with the recorded alternating reference; evaluation repeats characterize fixed checkpoints rather than independent learning runs.

Table 12: Independent comparison of update schedules at the same proposal and optimizer-update counts. Each condition uses 14 harness proposals and 30 RL updates, and is evaluated on the same 70 Test tasks through repeated evaluations at temperature zero. H denotes harness proposals; RL denotes solver updates.

Condition	Update order	Test accuracy (%)
One-shot sequential	H14 → RL30	54.29
Alternating	H7 → RL15 → H7 → RL15	**71.0**

D.4 Learning Trajectories and Training Diagnostics

Implementation records. The original temperature-zero trajectory comprises two Harness/RL cycles. Each RL phase inherits the model and program selected at the preceding boundary. Source paths, model and harness identities, and evaluation receipts are retained with the plotting data for reproducibility.

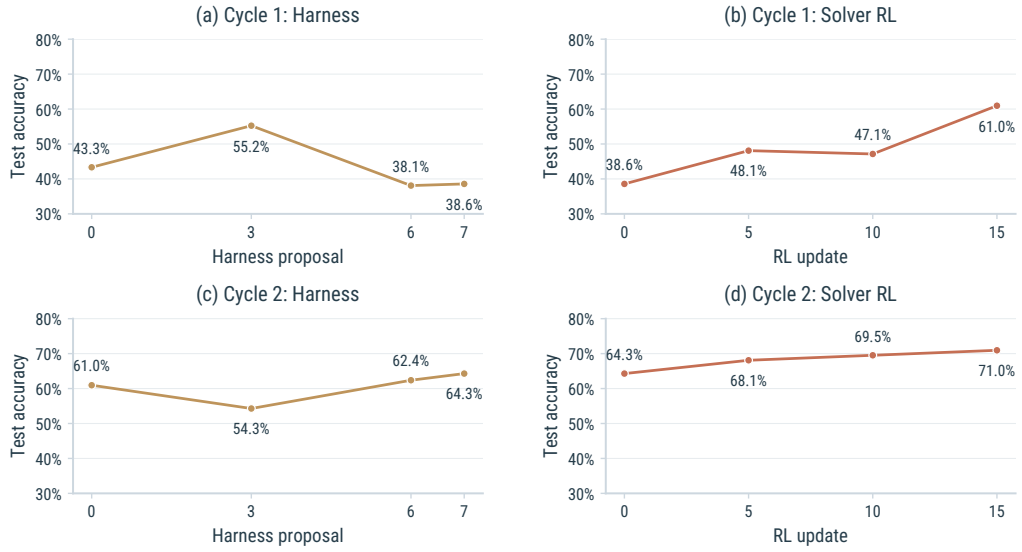


Figure 5: **Recorded stage checkpoints.** Panels follow H1, R1, H2, and R2 in learning order. Points show the repeated temperature-zero evaluations at stage boundaries and intermediate steps; the final R2 endpoint is 71.0%, as reported in Table 1.

Training and evaluation diagnostics. Figure 6 summarizes the 30-update training schedule. Train reward and solved-in-eight coverage describe changing training batches, not fixed Test performance. The mixed-reward-group fraction describes the availability of within-group learning signals. Train metrics summarize sampled attempts; sampled and optimizer-used attempts need not coincide.

Accuracy across scientific tasks. Figure 7 compares the four main conditions by task category using the category entries in Table 1.

Fixed-solver harness evolution. Figure 8 shows the 30-proposal Harness-only search with M0 fixed. On the temperature-zero, seven-category Test panel, accuracy is 43.3% at H0 and 57.14% after six proposals. Proposal 6 is the first measured positive checkpoint, with evaluations performed every three proposals. This provides a concrete instance of procedural improvement before model retraining. Test scores do not determine adoption.

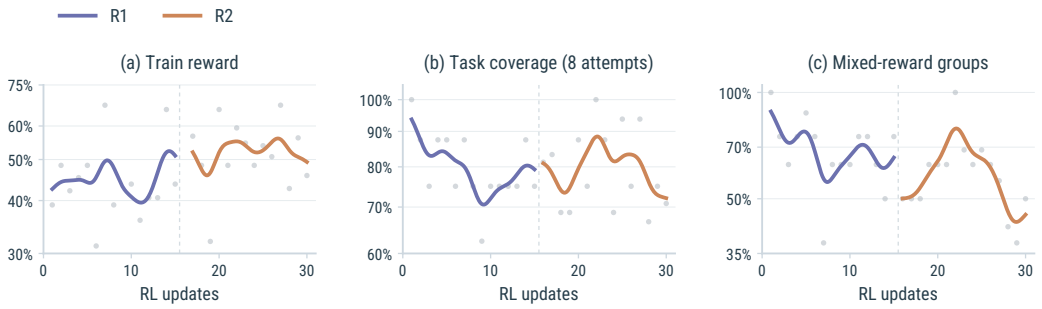


Figure 6: **Training outcomes and learning signals in R1 and R2.** Reward, task coverage over eight attempts, and mixed-reward-group fraction across 30 updates. Focused logarithmic y-axes show gray raw measurements and phase-wise Gaussian-smoothed curves ($\sigma = 1$ update).



Figure 7: **Category accuracy across the main comparison conditions.** H0, Harness-only, RL-only, and R2 use a fixed 0–100% color scale following Table 1.

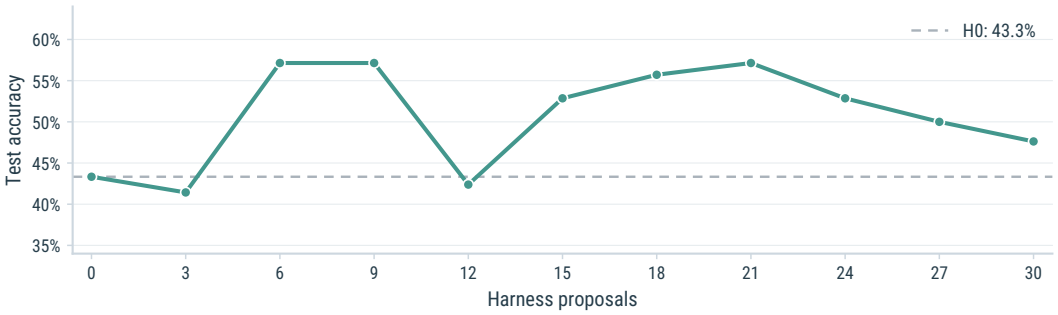


Figure 8: **Fixed-solver Harness-only trajectory.** Qwen3.5-4B remains fixed across 30 harness proposals, each following 16 Train interactions. Points show mean first-submission accuracy on the same seven-category Test panel through repeated evaluations at temperature zero, measured at H0 and after every third proposal. The dashed line marks H0; proposal adoption uses preflight, not Test scores.

D.5 Answer Delivery and Execution Behavior

Figure 9 summarizes answer outcomes for the initial and final systems. Correctness rises from 43.3% to 71.0%, while the unsubmitted fraction falls from 21.9% to 3.8%. Figure 10 complements these outcomes with model and tool calls, output-token usage, and correct answers per 1,000 output tokens.

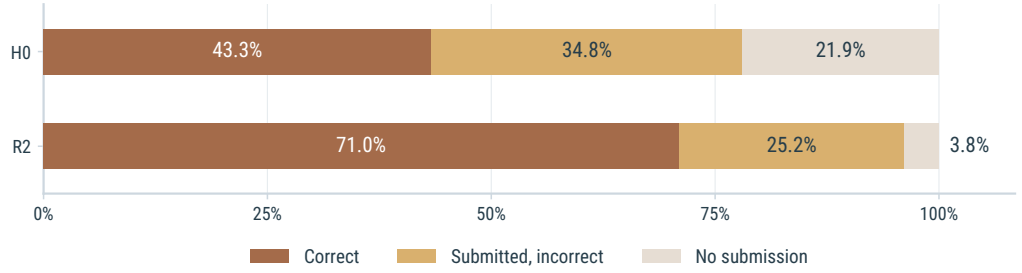


Figure 9: **Answer outcomes at H0 and R2.** Bars show the proportions of correct, submitted-but-incorrect, and unsubmitted attempts in the main comparison.



Figure 10: **Execution behavior at H0 and R2.** Submission, truncation stops, model/tool calls, and generated-token usage include failed attempts. Correct answers per 1,000 output tokens is a descriptive deployment statistic, not total compute, adaptation cost, or monetary cost.

E Auxiliary-Model Usage

Table 13 reports the auxiliary-model usage of full-program and prompt/skill adaptation with user replies. Totals include recoverable usage from failed API attempts. Cached and reasoning tokens are subsets of the recorded input and output totals, respectively. The table reports token usage rather than provider charges.

Table 13: Auxiliary usage for the full-program and prompt/skill conditions with user replies (Table 7), including recoverable usage from failed API attempts. Input/output totals are in millions of tokens; cached and reasoning tokens are subsets and are not added twice. Counts measure resource use, not a performance ranking.

Condition	HTTP attempts	Input (M)	Output (M)	Total (M)
User reply, full program	201	1.498	0.736	2.234
User reply, prompt/skill	182	0.759	0.520	1.279

F Case Studies and Interaction Evidence

F.1 From User Feedback to Reusable Procedures

We select two cases from the independent fixed-solver diagnostic run in Section D. Each connects a Train interaction to an adopted harness revision and a successful lookup on a different Test question. The solver remains Qwen3.5-4B throughout; the evolved harness is evaluated after seven proposals, before any model update. Both Test questions change from consistently incorrect to consistently correct across the recorded evaluation repeats, with no user feedback at Test time. The panels show one recorded evaluation attempt. These are comparisons of the complete harnesses; individual edits are not isolated.

Learning episode: user feedback becomes a reusable GMT reading rule

Train task. Find the gene whose promoter has a ZFP91 binding site in the GTRD gene-set file. The agent treats ZFP91 appearing inside another factor's gene set as support for its answer.

User reply (excerpt). *"In a GMT like 'c3.tft.gtrd...', the set name is the TF and the members are its target genes"*; *"Can you actually print that ZFP91 set and show which of the four candidates it contains"*.

Adopted revision (proposal 2). The improver cites this reply and adds a general GMT rule: field 1 is the set name, field 2 the description, and fields 3 onward the members. A harness-run probe prints the real file layout before the first model turn.

New evaluation task: query a different transcription factor

Which gene has a ZNF436 binding site in its promoter according to GTRD?
Options: A. SH2D3A B. BIRC7 C. GALNTL5 D. CCDC14.

Initial harness: incorrect parsing exhausts the tool budget

Execution. The agent finds the `ZNF436_TARGET_GENES` row, but then scans the *following 20 rows* as though each were one of that factor's target genes. Repeated large outputs are truncated.

Outcome. Nine model calls and nine tool calls; no answer submitted. This failure occurs in every recorded evaluation repeat.

Evolved harness: correct membership lookup and verified answer

Procedure. The harness supplies the GMT layout. The agent identifies the ZNF436 set and checks the four candidates in that row's member fields.

Tool output (excerpt):

```
Set:  ZNF436_TARGET_GENES
Is SH2D3A in members?  True
Is BIRC7 in members?  False
Is GALNTL5 in members?  False
Is CCDC14 in members?  False
```

Final answer: `<answer>A</answer>` — verified correct.
Outcome. Three model calls and three tool calls, including the harness probe; correct in every recorded evaluation repeat, without a user correction.

Figure 11: **Case 1: A gene-set reading lesson transfers to another lookup.** Feedback about ZFP91 is incorporated into the harness; the same fixed solver then retrieves the answer for ZNF436. Task wording and execution are summarized; quoted feedback and tool-output excerpts retain the recorded wording.

Learning episode: a user identifies the wrong JSON path

Train task. Compare ClinVar annotations for G1992A, L1308F, S1103Y, and A735E. The agent looks for `summary_records` inside `clinvar`, gets a `KeyError`, and treats the failed lookup as missing annotation.

User reply (excerpt). *"summary_records sits alongside clinvar, not under it. That's a fixable path error, not evidence of missing annotation."*

Adopted revision (proposal 7). Using the accompanying `KeyError` and container-type errors, the improver extends the existing file probe to print nested paths, node types, and child keys. Later lookups receive the actual hierarchy before generating code.

New evaluation task: distinguish benign from pathogenic variants

Which variant is most likely to be benign according to the local ClinVar snapshot?
Options: A. L1256P B. N414H C. P1476L D. H1435Y.

Initial harness: failed extraction is mistaken for absent evidence

Execution. The agent first calls a dictionary method on a string. Later queries print an empty benign-variant list or no output.

Agent statement (excerpt). *"The output is empty again, which means N414H, P1476L, and H1435Y are not in the snapshot."*

Outcome. The illustrated attempt reaches nine tool calls without an answer. Across the recorded evaluation repeats, attempts either fail to submit or submit an incorrect answer; none is correct.

Evolved harness: navigate the nested records and compare annotations

Harness-provided structure (condensed).
`candidate_records[0]` contains `summary_records`; each summary contains `germline_classification`.

Executed lookup (excerpt). `summary = rec['summary_records'][0]`
The agent matches each candidate's `protein_mutation` and reads its classification from that summary.

Retrieved classifications (condensed):

L1256P	Likely pathogenic
N414H	Likely benign
P1476L	Pathogenic
H1435Y	Pathogenic

Final answer: `<answer>B</answer>` — verified correct.
Outcome. Four model calls and four tool calls, including the harness probe; correct in every recorded evaluation repeat, without a user correction.

Figure 12: **Case 2: A structure-aware harness enables an evidence-based variant comparison.** User feedback and execution errors identify a nesting mistake in a Train interaction. The evolved harness exposes the hierarchy on a different Test question, where the fixed solver retrieves the classifications and submits the verified answer. The original protein sequence is omitted from the task summary.

G Comparison with Recent Learning Systems

The comparison concerns the object being updated, its learning signal, and the empirical question addressed here. It does not imply that a listed approach lacks every control used in this paper. No cross-paper performance ranking is made, because task distributions, backbones, budgets, and evaluation protocols differ.

Table 14: Positioning relative to closely related work. The last column identifies the corresponding focus of our study; it does not assert that a feature is absent from every other approach.

Work	Established approach	Focus of this study
Co-Scientist / EvoScientist	Hypothesis evolution and persistent research experience (Gottweis et al., 2026; Lyu et al., 2026)	Executable scientific tasks with fixed verifiers and model RL
DGM / Hyperagents / AIDE[2]	Agent-code and improvement-process evolution (Zhang et al., 2025a; 2026b; Srikanth et al., 2026)	Fixed auxiliary improver; execution/training/order decomposition
Co-Harness	Validated local harness patches alternating with filtered-trajectory SFT (Chen et al., 2026b)	Full-program revisions, preflight adoption, fresh-rollout GRPO
EvoHarness-RL	Learned construction and access of external harness state (Ning et al., 2026)	Alternating program revisions and solver updates
CoEvolve / RLAnything	Adaptive task data, environments, or rewards (Yang et al., 2026; Wang et al., 2026b)	Tasks, environments, and independent verifiers are given
OpenClaw-RL / LOFA	Interaction signals train the policy (Wang et al., 2026a; Zhang et al., 2026a)	Replies guide programs; independent checks reward RL
MetaClaw	Skill evolution and policy optimization (Xia et al., 2026)	Full executable edits versus prompt/skill edits
SIA	Feedback-agent-selected harness and weight updates (Hebbar et al., 2026)	Fixed update recipe with crossed and schedule controls
HELIX	Traceable harness evolution and verified learning records (Fan & Huang, 2026)	Completed model updates and two-cycle schedule controls

H Researcher Collaboration and the Scientific Workspace

These qualitative cases illustrate researcher objectives, feedback, and inspection. They are separate from the simulated-user benchmark experiments and are not evidence that these researcher sessions were used for training or produced subsequent model improvements.

H.1 Researcher Interaction and Task Requirements

Setup. We examine two real researcher interactions with the scientific workspace. Requests, supplied materials, agent responses, and subsequent researcher input support a qualitative assessment of scientific assistance and opportunities for reinforcement learning (RL) task construction. The following descriptions summarize the recorded interactions; their derived objectives illustrate the task interface rather than a completed learning evaluation.

Refining a JAK1 investigation. A researcher asked ScienceBuddy to design a study of JAK1, immunotherapy outcomes, and the immune microenvironment in small-cell lung cancer using public single-cell transcriptomes and IMpower133 bulk RNA data. In response to the scope refinement, ScienceBuddy organized a gene-specific plan with treatment-by-JAK1 interaction tests, patient-level expression summaries within cell types, and immune-state signatures. The plan assigned Seurat/Scanpy to single-cell analysis, UCell/AUCell to signature scoring, and CellChat/NicheNet to subsequent cell-communication analyses. This plan distinguished treatment-effect modification from prognosis and prioritized mechanistic follow-up.

Connecting evidence in an ARL4C study. A researcher requested a presentation connecting the background and results of an ARL4C study, then specified panel selection, conclusions, mechanism schematics, and speaker notes. Using text and figure captions organized through Python/PyPDF2, ScienceBuddy linked candidate screening to cellular and molecular evidence. It highlighted depletion and conditional knockout comparisons for cellular attribution, blockade for functional depen-

dence, and kinetic and rescue assays for molecular interpretation. Panel-selection rationales and notes linked each scientific claim to its supporting comparison.

From requests to task specifications. These cases illustrate how researcher requirements translate into task objectives, evaluation criteria, and required artifacts (Figure 13). The JAK1 refinement yields a study-planning objective whose criteria preserve gene-specific scope and place association analyses before mechanistic follow-up. The ARL4C request yields a presentation objective whose criteria link claims to supporting panels and comparisons, with conclusions and speaker notes accompanying the slide outline. Such task specifications motivate the replayable-task interface in Section 2.2. Converting these researcher cases into validated scoring rules and training environments remains separate from the benchmark experiments reported here.

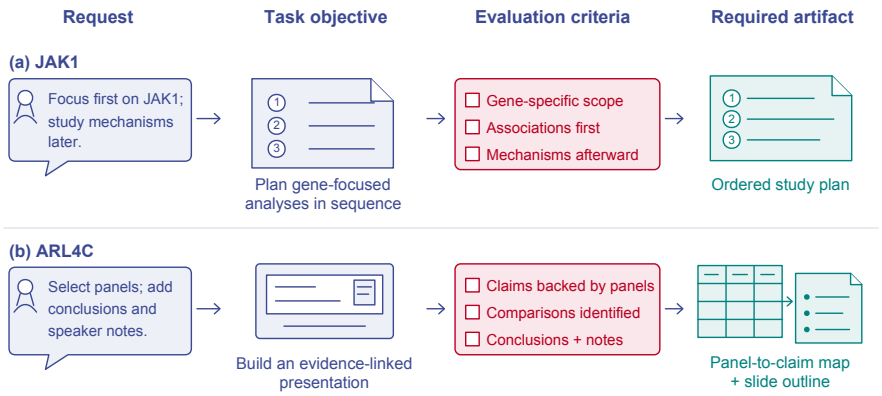


Figure 13: **From researcher requests to task specifications.** (a) A JAK1 scope refinement defines an ordered, gene-focused study plan. (b) ARL4C presentation requirements define an evidence-linked presentation and panel-to-claim map. Requests are translated and abridged from real interactions; the task objectives, evaluation criteria, and required artifacts are illustrative derivations, not archived rubric packages or scored outputs.

H.2 Scientific Workspace and User Experience

Scientific scope. ScienceBuddy combines multimodal input, tool-supported analysis, and researcher interaction within a shared scientific workspace. Its document handling and execution interfaces support multiple scientific domains, while the current tools and data specialize in biomedicine. The following recorded session illustrates how researchers connect visual scientific material to target analysis, evidence retrieval, and further questions.

Multimodal input and evidence inspection. Researchers can supply documents, tables, biological sequences, and images alongside natural-language requests. In Figure 14, an uploaded immune-signaling diagram guides the identification of molecular targets and the organization of related drug and pathway knowledge. The response connects visual entities to an evidence table, distinguishing a retrieved PDE4/rolipram fragment from CD40 and AHR searches that returned no matches. The conversation, input composer, and Compute panel bring the scientific material, response, and execution history into one inspectable view. The panels use anonymized English interface reconstructions from the recorded session.

Researcher inspection through interface controls. Researcher interaction also includes navigation and inspection actions beyond conversational input (Figure 15). In the demonstration, the researcher opens an uploaded diagram at a larger scale, switches from Chat to Trajectory, and selects a tool event to inspect its metadata, input, and output. The selected UniProt event exposes an earlier HMGCR lookup while later requests remain in the same session. These controls let the researcher examine source material, follow the execution history, and revisit the basis of a response without starting a new conversation.

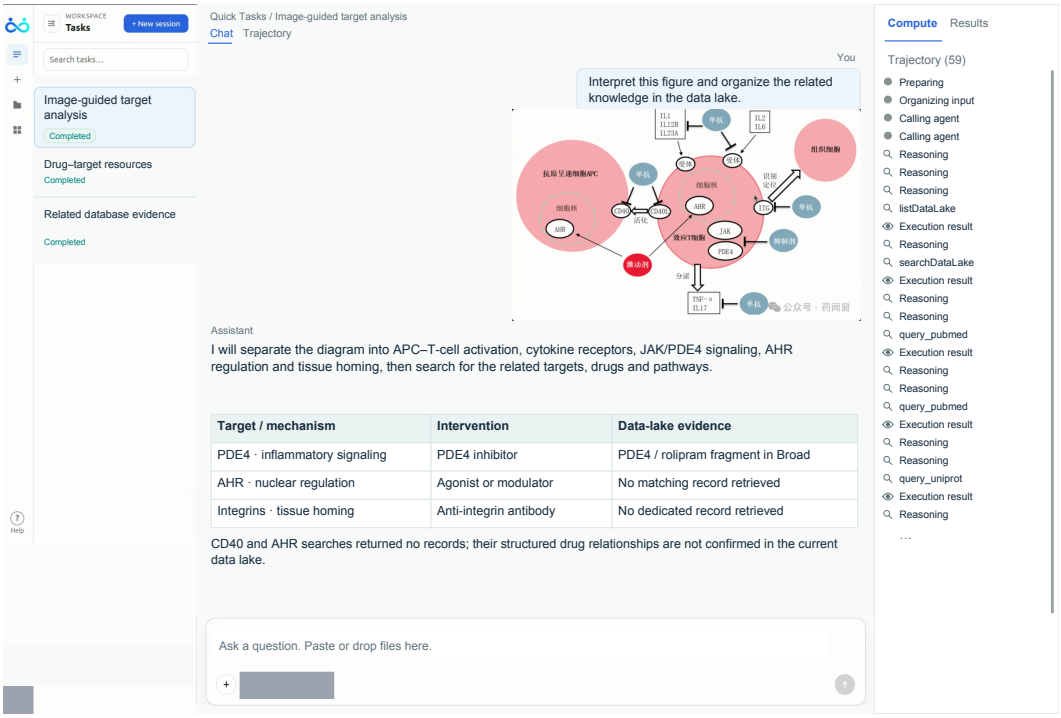


Figure 14: **A workspace for multimodal scientific analysis.** A researcher supplies a scientific diagram and requests related knowledge. The Chat view connects visual interpretation to a structured target–evidence table, while the Compute panel exposes execution records. Retrieved evidence and gaps in the available data remain visible for researcher inspection. UI text and dialogue are reconstructed in English from the recording; uploaded figures retain their original appearance and language. Account and model identifiers are masked.

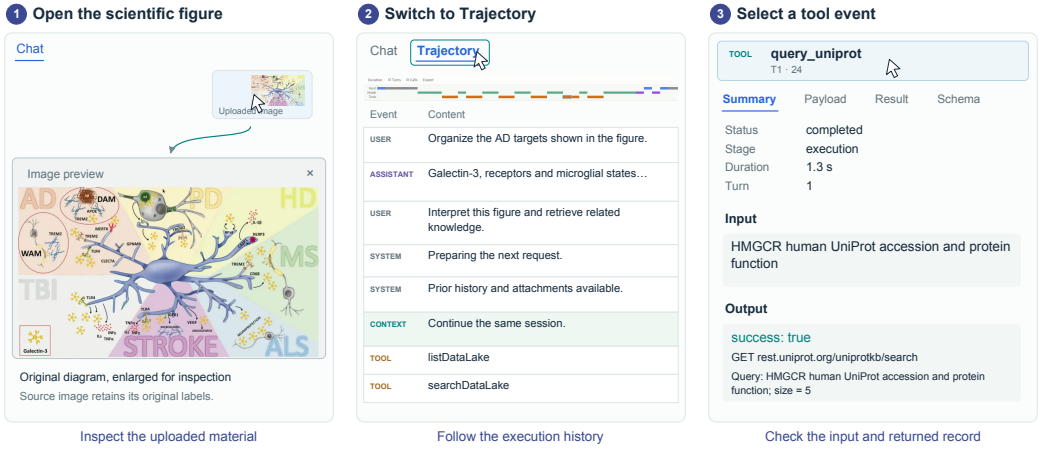


Figure 15: **Researcher inspection beyond the conversation.** Opening an uploaded image reveals its scientific details; switching to Trajectory exposes the execution history; selecting a tool event opens its input, output, and metadata. The example revisits an earlier HMGCR protein lookup within the continuing session. English interface reconstructions highlight controls used in the recording; the uploaded diagram and timeline retain source pixels. Cursor markers indicate the inspected controls.