CoEvoBench: Can LLMs Drive Recursive Co-Evolution of Agentic Systems?

Anonymous authors
Paper under double-blind review

Abstract

Autonomously improving agentic systems requires researchers to translate execution feedback into effective updates to model weights and the harness governing interaction with tools and environments. Existing evaluations primarily focus on model training or harness optimization, leaving research capability under joint access to both intervention types insufficiently assessed. We introduce **CoEvoBench**, a benchmark that evaluates LLM researchers on autonomous model–harness improvement across five agentic domains. Researchers start from a common initialization within each domain and independently choose training, harness revisions, and their combinations within a bounded resource budget, using development feedback to guide further research. The submitted model–harness pair then executes independently on held-out tasks to measure the system improvement delivered by the researcher. Across five frontier researcher models, we observe substantial but uneven gains across domains, with no researcher consistently achieving the strongest improvements on every benchmark. Research gains concentrate in key breakthroughs: among improving runs with at least three selection-set candidate evaluations, the largest increase in best-so-far performance accounts for a median of 62.5% of the total observed gain. These results highlight the challenge of extending isolated breakthroughs into sustained improvement. CoEvoBench provides a framework for evaluating this capability.

1 Introduction

At the heart of recursive self-improvement (RSI) is the ability to turn feedback from one round of operation into capability gains in the next. In agents deployed for real-world applications, this process takes a concrete, observable form: software development, application operation, and service interactions produce sequences of tool calls, environment responses, and execution traces that reveal where a system fails and whether revisions yield progress. Performance depends on both model decisions and the harness governing execution, creating opportunities to improve training data, model weights, context organization, and tool interaction. Developing agentic systems requires substantial manual iteration, motivating automated approaches to system improvement. Having an LLM researcher analyze this evidence, implement changes, and evaluate the resulting system provides a controlled setting for studying the feedback-driven research capabilities required for RSI.

Existing work has begun to evaluate and automate this improvement process, with different emphases on task settings and intervention scope. PostTrainBench (Rank et al., 2026) evaluates model improvement through end-to-end post-training, focusing on reasoning tasks and weight updates, while optimization of interactive execution mechanisms remains to be examined. RSIBench-Data (Meng et al., 2026) focuses on data-centric research, whereas agent systems also offer opportunities to improve tool interfaces, context organization, and execution control. DarwinX (Zhang et al., 2026b) explores harness evolution with a fixed model, leaving interactions between model training and harness changes outside its scope. ScienceBuddy (Xue et al., 2026b) brings both types of updates into a shared improvement loop, demonstrating the feasibility of co-evolution; systematic evaluation is still needed to establish whether LLM researchers can autonomously advance such loops across task domains and sustain gains using feedback from the updated system.

Joint access to model training and harness modification enables evaluation of recursive improvement at the system level. Researchers form improvement hypotheses from execution feedback, implement

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.

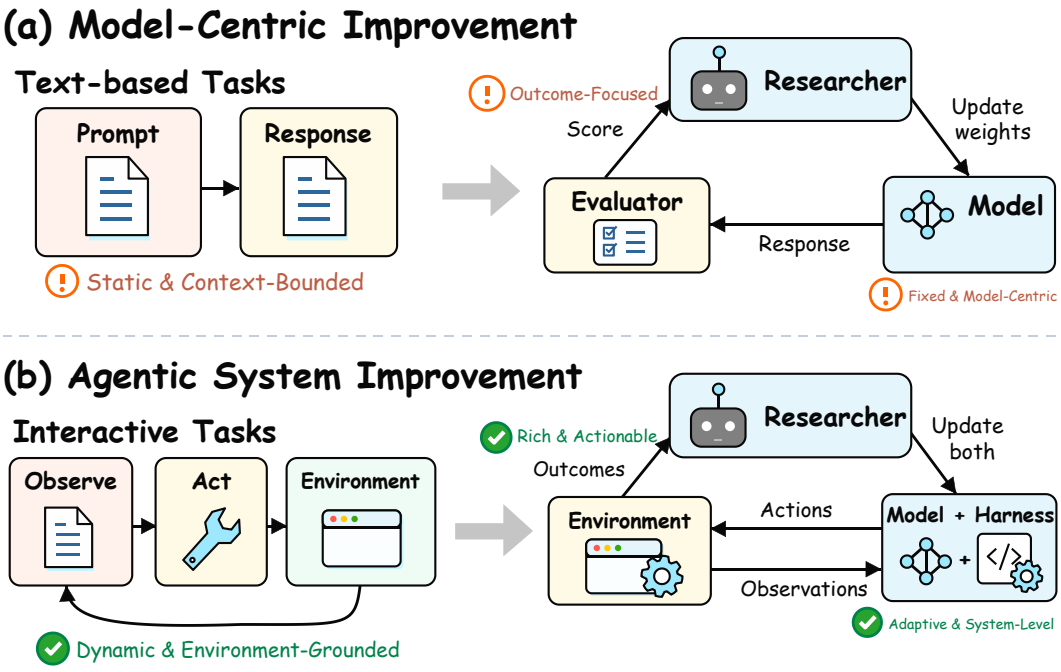

Figure 1: **Motivation for CoEvoBench. (a) Model-centric improvement:** for reasoning tasks, the researcher uses task feedback to update model weights and improve reasoning. **(b) Agentic-system-centric improvement:** for agentic tasks, CoEvoBench evaluates researchers that use feedback from task execution and environment interaction to improve model weights, the harness, or both. The harness governs how the model interacts with tools and the environment.

and evaluate updates, and use the revised system and its new feedback as the starting point for the next round of research. As this loop unfolds, system capabilities and failure modes evolve, requiring researchers to adapt their research directions. Our central question is: **Can LLM researchers drive this recursive improvement process and achieve further system gains through successive iterations?** Evaluating this capability requires examining both the final delivered performance and the within-run development process. Figure 1 illustrates this extension from model-centric optimization to interactive system improvement.

We introduce **CoEvoBench**, a benchmark for evaluating LLM researchers on driving the recursive improvement of model–harness systems. It spans five agentic task domains: software repair, terminal operation, application interaction, interactive service workflows, and multi-turn tool use and memory. Researchers start from a common base model and initial harness within each domain, autonomously train models and revise harnesses within a bounded resource budget, and iterate on candidate systems using development feedback. The platform fixes task environments and scoring rules, and evaluates the submitted model–harness pair through independent execution on held-out tasks. By combining final system performance with development trajectories, CoEvoBench examines both the improvement researchers deliver and how gains emerge during development.

Our evaluation of five frontier researcher models shows that research gains concentrate in key breakthroughs. Researchers achieve substantial improvements through model training, harness revisions, and their combinations. Among improving runs with at least three selection-set candidate evaluations, the largest increase in best-so-far performance accounts for a median of 62.5% of the total observed gain. This concentrated pattern of progress highlights a central challenge for recursive improvement: turning subsequent execution feedback into the next effective breakthrough. Figure 2 summarizes the mean held-out performance across the five domains, alongside the initial Base system and the Instruct model with a reference harness.

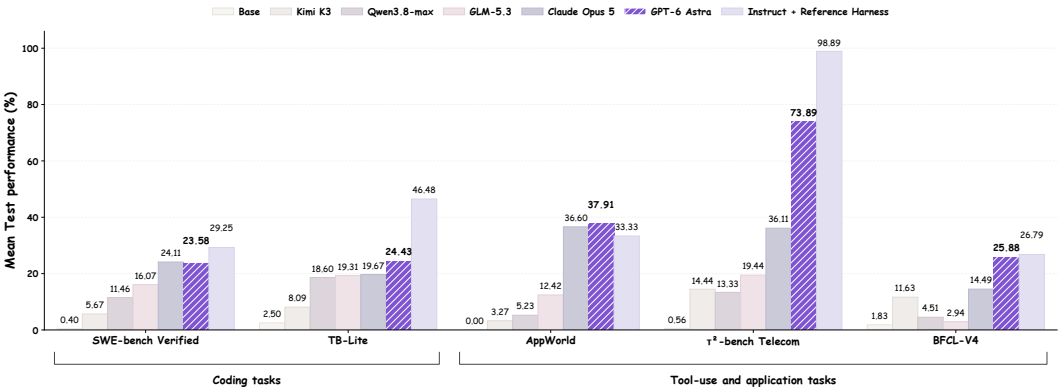

Figure 2: **CoEvoBench main results.** Mean test performance (%) of systems developed by each researcher across five agentic benchmarks, averaged over three independent development runs per researcher–benchmark pair. Base denotes Qwen3.5-9B-Base with the initial harness; Instruct + Reference Harness denotes Qwen3.5-9B-Instruct with a reference harness. Researcher names identify the models developing the systems.

Our main contributions are:

- **Introducing a benchmark for system-level recursive improvement.** CoEvoBench jointly exposes model training and harness modification across five agentic domains, with a shared development protocol and held-out evaluation.
- **Characterizing the capabilities and bottlenecks of autonomous improvement.** Evaluating five frontier researchers reveals substantial but uneven system gains concentrated in individual breakthroughs, highlighting the challenge of sustaining progress.
- **Uncovering model–harness interdependence.** Through paired evaluations and development case studies, we reveal how the evolving harness changes the effectiveness of model updates: harness revisions can reverse training gains, while subsequent training on refreshed trajectories can restore them.

2 Related Work

Automated AI research and evaluation Automated systems conduct scientific research (Lu et al., 2024), while benchmarks assess ML experimentation, engineering, expert research, and paper replication (Huang et al., 2024; Chan et al., 2025; Wijk et al., 2025; Starace et al., 2025). Recent suites evaluate post-training, data research, algorithm design, and agentic RL (Rank et al., 2026; Meng et al., 2026; Chi et al., 2026; Chen et al., 2026c), alongside agent construction under business and serving constraints (Shi et al., 2026).

Harness-level agent improvement With fixed model weights, agents improve through feedback and reusable skills (Shinn et al., 2023; Wang et al., 2023), prompt and program search (Agrawal et al., 2026; Hu et al., 2025), and evolving control logic and context management (Zhang et al., 2026a; Lee et al., 2026; Zhang et al., 2026b; Wu et al., 2026). Benchmarks also assess transfer from retained experience and skill libraries to later tasks (Xue et al., 2026a; Guan et al., 2026).

Agent training and model–harness co-evolution Agent training uses trajectory tuning and executable environments (Zeng et al., 2023; Pan et al., 2025), with harnessed RL preserving deployment interactions (He et al., 2026). Co-evolution combines feedback-directed intervention and harness-conditioned training (Hebbar et al., 2026; Chen et al., 2026d;a;b), scientific-agent learning and operator composition (Xue et al., 2026b; Tan et al., 2026), and expert correction for model–harness compatibility (Yu et al., 2026).

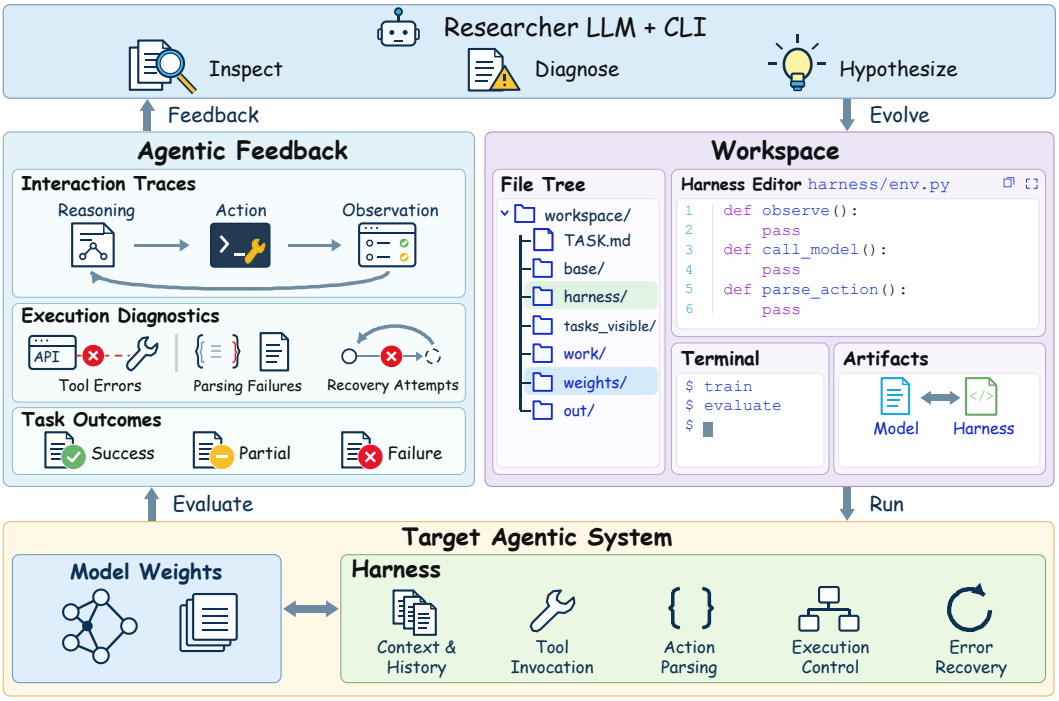

Figure 3: **An overview of the CoEvoBench development loop.** An LLM researcher improves a target agent consisting of model weights and an executable harness. At each iteration, the researcher uses feedback to diagnose failures, formulate improvement hypotheses, and modify the model weights or the harness. The resulting target agent executes development tasks, producing outcomes and interaction traces. Visible tasks provide task details and execution traces for analysis, while selection-set evaluations return performance feedback without exposing task contents or trajectories. These observations guide the researcher's next modifications, closing the modification–execution–feedback loop. Within a 24-hour development budget, the researcher nominates a final model–harness pair, which is evaluated on held-out test tasks without further modification.

3 CoEvoBench

CoEvoBench evaluates the ability of a researcher R to improve a target agent using development feedback. We represent the target agent as $S = (\theta, H)$, where θ denotes model weights and H is the executable harness governing tool use and environment interaction. For domain b, a development run is specified by

$$\mathcal{P}_b = \left(S_b^{(0)}, \mathcal{I}_b, \mathcal{U}_b, \mathcal{E}_b^{\text{dev}}, \mathcal{B}\right), \tag{1}$$

where $S_b^{(0)} = (\theta_0, H_b^{(0)})$ is the initial system, $\mathcal{I}_b$ contains materials accessible to the researcher, $\mathcal{U}_b$ is the permitted action space, $\mathcal{E}_b^{\text{dev}}$ provides development evaluation feedback, and $\mathcal{B}$ is the resource budget. The researcher iterates within this setup and submits a final system $S_b^\star = (\theta_b^\star, H_b^\star)$, whose improvement is measured through independent testing. Figure 3 connects researcher-driven modifications, target-agent execution, and feedback within the development loop.

3.1 Benchmark Suite

The domain set $\mathcal{D}$ comprises five agentic benchmarks: SWE-bench Verified (Chowdhury et al., 2024), TB-Lite (OpenThoughts-Agent Team et al., 2026), AppWorld easy (Trivedi et al., 2024), the τ^2-bench Telecom base subset (Barres et al., 2025), and BFCL-V4 multi-turn and memory tasks (Patil et al., 2025; Mao et al., 2025). They cover software repair, terminal operation, application interaction, service interaction, and multi-turn function calling and memory, respectively.

For each domain $b \in \mathcal{D}$, we partition the task pool D_b into three disjoint sets:

$$D_b = D_b^v \dot{\cup} D_b^s \dot{\cup} D_b^t. \tag{2}$$

We target a 1:2:2 split, with domain-specific adjustments reflecting task characteristics (Appendix C.1). The visible set D_b^v provides material for failure analysis and development; the selection set D_b^s allows researchers to assess how well their modifications generalize to unseen tasks; and the test set D_b^t supports final evaluation. Accessible materials $\mathcal{I}_b$ include initial instructions, system and tool documentation, and visible-set task materials.

We stratify SWE-bench Verified by repository and TB-Lite by difficulty. For AppWorld, we assign each scene's variants to the same split and stratify scenes by their normal or challenge source. Telecom is stratified by operation type. For BFCL, we group multi-turn variants by task family and keep each memory scenario, its cross-backend QA records, and prerequisite conversations in one split.

3.2 Research Setup and Action Space

All runs start from Qwen3.5-9B-Base weights θ_0 and a domain-specific initial harness $H_b^{(0)}$. The resource budget $\mathcal{B}$ comprises 24 hours of development, four NVIDIA H20 GPUs, and a $100 teacher-model calling budget. Researchers allocate GPUs to training and inference.

The action space comprises inspection and evaluation, model updates, and harness modifications:

$$\mathcal{U}_b = \mathcal{U}_b^{\text{inspect/eval}} \cup \mathcal{U}_b^{\text{model}} \cup \mathcal{U}_b^{\text{harness}}. \tag{3}$$

Inspection and evaluation actions support artifact inspection, execution-feedback analysis, and candidate evaluation. Model-update actions include constructing training data, querying the teacher, and updating weights θ. Harness modifications act on H, including tool interfaces, execution control, context management, and memory mechanisms. Feedback guides action selection and ordering.

Agentic tasks involve multi-step tool and environment interactions, making successful trajectories difficult to obtain from the initial system. We therefore allow limited access to a fixed teacher model, Qwen-3.8-27B, for constructing training data through distillation. Teacher assistance is available as a data-construction tool within the model-update action space.

3.3 Development Protocol

During development, researchers can inspect task details and execution trajectories on D_b^v to diagnose failures, formulate improvement hypotheses, and construct training data. The development-feedback operator $\mathcal{E}_b^{\text{dev}}$ supports visible- and selection-set evaluations. Evaluations on D_b^s return only performance feedback to assess generalization to unseen tasks.

At development step t, the researcher selects an action $a_t \in \mathcal{U}_b$ based on the interaction history $\mathcal{T}_t$, receives an observation o_{t+1}, and incorporates it into subsequent decisions:

$$a_t \sim R(\cdot \mid \mathcal{T}_t), \qquad \mathcal{T}_{t+1} = \mathcal{T}_t \oplus (a_t, o_{t+1}). \tag{4}$$

Here, $\mathcal{T}_0$ consists of the initial accessible materials $\mathcal{I}_b$, and $\oplus$ appends the action and returned observation to the history. Through this loop, the researcher inspects feedback, modifies the system, and evaluates candidates to develop a final solution.

The researcher nominates a final system $S_b^\star = (\theta_b^\star, H_b^\star)$ within the 24-hour development budget, with early termination permitted. After submission, the system executes independently on the test set without further modification.

CoEvoBench's experiment execution and evaluation infrastructure uses sandbox isolation and benchmark-data access restrictions during development. Researchers modify models and harnesses in isolated workspaces with network access enabled, while datasets that directly expose benchmark task data are blocked. Visible-set materials are supplied through the platform. Selection and test data remain protected, and selection-set task contents and execution trajectories are withheld.

3.4 Evaluation and Scoring

The platform evaluates the final and initial systems on D_b^t under the same domain-specific scoring protocol. We measure complete-task success for SWE-bench Verified, mean task reward for TB-

Lite, complete-task success for AppWorld and Telecom, and a weighted combination of multi-turn and memory accuracy for BFCL, with weights of 60% and 40%, respectively.

After each experiment, we manually audit researcher trajectories for unauthorized access to evaluation data or manipulation of evaluation results. For runs with confirmed cheating, we re-evaluate the base model paired with the final submitted harness. Let $\mathcal{E}_b^{\text{test}}(S)$ denote the test score of system S on domain b. The reported score is

$$\widehat{\mathcal{E}}_b(S_b^\star) = \begin{cases} \mathcal{E}_b^{\text{test}}((\theta_0, H_b^\star)), & \text{if cheating is confirmed,} \\ \mathcal{E}_b^{\text{test}}((\theta_b^\star, H_b^\star)), & \text{otherwise.} \end{cases} \tag{5}$$

The improvement delivered by the researcher over the initial system is

$$J_b(R) = \widehat{\mathcal{E}}_b(S_b^\star) - \mathcal{E}_b^{\text{test}}(S_b^{(0)}). \tag{6}$$

We report per-domain test performance and improvements, and summarize cross-domain performance using normalized weights. Let $\bar{s}_b(R)$ denote the mean reported test score across three independent development runs of researcher R on domain b, B_b the test score of the initial model paired with the initial harness, and I_b the test score of the instruction-tuned model paired with a reference harness. Using the baseline gap $I_b - B_b$ as the reference scale for each domain, we define

$$w_b = \frac{(I_b - B_b)^{-1}}{\sum_{j \in \mathcal{D}} (I_j - B_j)^{-1}}, \qquad \text{Avg.}(R) = \sum_{b \in \mathcal{D}} w_b \bar{s}_b(R). \tag{7}$$

The weights sum to one and are fixed across researchers. With this weighting, improvements that close the same fraction of the baseline gap contribute equally to the aggregate score.

4 Experiments

4.1 Experimental Setup

Researcher configurations. We evaluate five frontier researcher models—GPT-6 Astra, Qwen3.8-max, GLM-5.3, Kimi-K3, and Claude Opus 5. GPT-6 Astra uses reasoning effort high and operates through Codex, while Qwen, GLM, Kimi, and Claude operate through Claude Code; each model–CLI pair constitutes a researcher configuration.

Target models and initial harnesses. All target agents are initialized from Qwen3.5-9B-Base and paired with a domain-specific initial harness. SWE-bench Verified and TB-Lite start from early versions of SWE-agent and Terminus, respectively. AppWorld uses a minimal code-generation harness, Telecom uses a lightweight text-based ReAct agent, and BFCL pairs its official execution loop with a lightweight policy layer.

Execution settings and baselines. All runs follow the resource budget and development protocol in Section 3. The initial-system baseline (Base) pairs Qwen3.5-9B-Base with each domain's initial harness. The Reference configuration pairs Qwen3.5-9B-Instruct with a domain-specific reference harness. We additionally evaluate the base model with the final harness to measure the gain from weight updates under the same harness. We report mean test performance across three independent development runs per researcher–benchmark pair and improvement over the initial system.

4.2 Main Results

Overall performance. Our evaluation reveals substantial but uneven system-level gains from recursive model–harness co-evolution. Table 1 summarizes the mean test performance across the five benchmarks. GPT-6 Astra leads on four benchmarks and ranks second on SWE-bench Verified. Claude Opus 5 leads SWE-bench Verified with 24.11%, compared with Astra's 23.58%. On AppWorld, Astra improves the initial system from 0.00% to 37.91%, surpassing the reference system built with Qwen3.5-9B-Instruct and a reference harness (33.33%). On TB-Lite, the strongest developed system achieves 24.43%, compared with 46.48% for the reference system. These results highlight substantial headroom for recursive co-evolution and suggest that its effectiveness depends on the interaction between the researcher and the target task.

Table 1: **Main results on CoEvoBench.** All target agents start from Qwen3.5-9B-Base paired with a domain-specific initial harness. Researchers can modify model weights and the harness within a 24-hour development budget using four NVIDIA H20 GPUs. Results report mean held-out test performance (%) over three independent development runs per researcher–benchmark pair.

Researcher	Avg.	SWE-bench Verified	TB-Lite	AppWorld	τ^2-bench Telecom	BFCL-V4
Base	1.09	0.40	2.50	0.00	0.56	1.83
Frontier models						
Kimi K3	7.93 +6.84	5.67 +5.27	8.09 +5.59	3.27 +3.27	14.44 +13.88	11.63 +9.80
Qwen3.8-max	9.40 +8.31	11.46 +11.06	18.60 +16.10	5.23 +5.23	13.33 +12.77	4.51 +2.68
GLM-5.3	12.24 +11.15	16.07 +15.67	19.31 +16.81	12.42 +12.42	19.44 +18.88	2.94 +1.11
Claude Opus 5	24.19 +23.10	**24.11** +23.71	19.67 +17.17	36.60 +36.60	36.11 +35.55	14.49 +12.66
GPT-6 Astra	**31.23** +30.14	23.58 +23.18	**24.43** +21.93	**37.91** +37.91	**73.89** +73.33	**25.88** +24.05
Instruct + Reference Harness	37.42	29.25	46.48	33.33	98.89	26.79

Per-domain performance. Researcher advantages are more task-dependent on demanding agentic coding tasks. On TB-Lite, Astra leads with a mean score of 24.43%, followed by Opus at 19.67% and GLM at 19.31%. This pattern is consistent with the diversity of effective research strategies in open-ended coding environments. Across the comparatively simpler tool-use and application tasks, Astra achieves the highest mean scores on all three benchmarks. On the Telecom subset of τ^2-bench, Astra improves the initial system from 0.56% to 73.89%, suggesting that reusable domain workflows offer productive opportunities for improvement. On BFCL-V4, Astra reaches 25.88%, compared with Opus 5's 14.49%, while balancing gains across different task components remains challenging. Its advantage is narrower on AppWorld, where Opus trails by only 1.31 percentage points, suggesting that application workflows offer improvement opportunities across researchers.

5 Analysis

5.1 Dynamics of Recursive Improvement

Gains concentrate in a single major jump. Among improving runs with at least three selection-set candidate evaluations, the largest increase in best-so-far performance accounts for a median of 62.5% of the total observed gain. Figure 4 illustrates this concentration of progress: major gains punctuate fluctuations in candidate performance, with subsequent exploration producing further gains, plateaus, and regressions.

Further exploration can plateau around a strong candidate. In one AppWorld run, Opus develops a harness for authentication, pagination, and context handling, reaching a mean selection score of 64.93%. The researcher then explores prompt revisions and self-distillation using 90 successful trajectories; these branches all score below the established candidate. Removing a pre-submission self-check ultimately yields 66.04%, with repeated evaluations placing its performance close to that of the earlier candidate. The main gains in this run arise during harness construction, followed by limited additional improvement from prompt and training exploration.

Turning feedback into further gains. The concentration of gains highlights the marginal returns to further research. Some runs achieve additional breakthroughs through subsequent adjustments, while others repeatedly explore around an established performance level. Turning further development effort into effective improvements remains a challenge for sustained recursive progress.

5.2 Task-Dependent Improvements and Component Coupling

Improvement strategies in agentic coding. In TB-Lite, researchers improve the base model's task execution by fixing command parsing, stopping behavior, and execution feedback, while model updates can impair these existing capabilities. In one Astra run, the base model scores 8.59% on the selection set under the same harness, compared with 4.25% after training. SWE-bench Verified also exhibits an effective route through coordinated training and interface design: after GLM aligns the

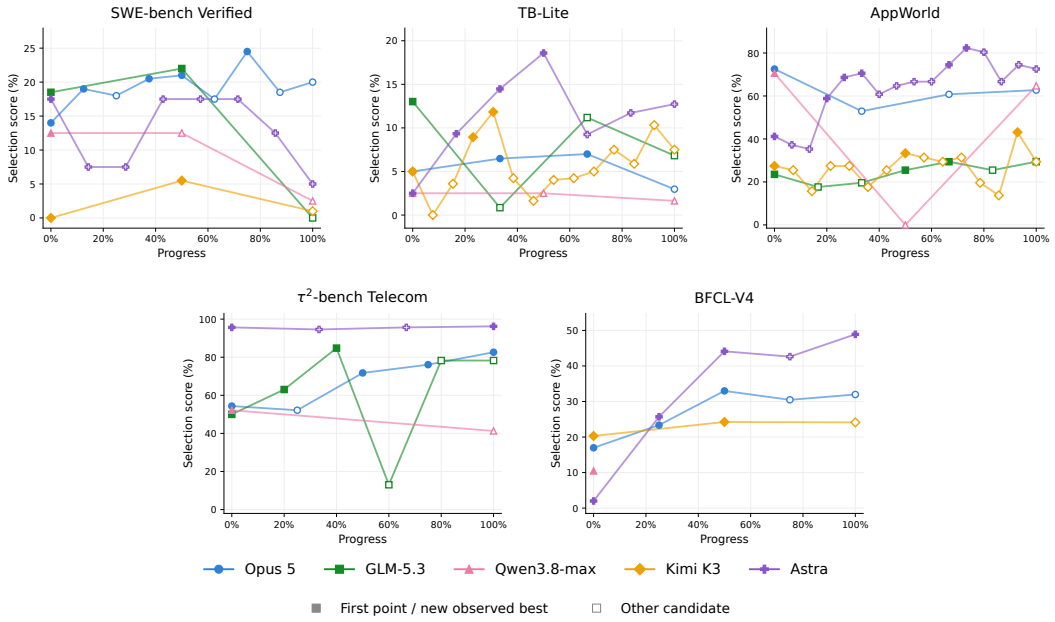

Figure 4: **Development trajectories across five benchmarks.** Each curve shows the run with the most iterations for a researcher–benchmark pair. Progress normalizes the candidate evaluation sequence from 0% (the first candidate) to 100% (the last). Filled markers indicate the first candidate or a new within-run best; hollow markers indicate other evaluated candidates.

execution interface with publicly available successful trajectories, the trained model solves 44/200 selection tasks, compared with 0/200 for the base model under the same interface. This contrast illustrates the role of training in adapting the model to a new interaction protocol.

Complementary gains in workflow-oriented tasks. AppWorld exhibits complementary benefits from execution support and behavioral learning: harnesses support authentication and interaction-history management, while training improves the model's execution of application workflows. In one Opus AppWorld run, harness revisions increase test success from 0/51 to 10/51 tasks, and adding training raises it further to 19/51. These results illustrate joint gains from model training and harness revisions. Configuration-level comparisons reveal positive and negative training gains, alongside improvements from combining base and trained weights (Appendix J).

Dynamic model–harness coupling. This relationship also evolves during development. In Astra's AppWorld run, harness revisions change the gain from the same model update from $+5$ to -4 solved selection tasks. Retraining on refreshed successful trajectories restores a gain of $+4$ tasks under the new harness (Figure 5). This reversal reveals the dynamics of joint optimization: harness improvements change the value of model updates, and subsequent training must adapt accordingly. Appendix H.2 reports the paired configurations and training details.

5.3 Research Strategies and Their Outcomes

Final performance depends strongly on the strategy pursued in each run. Across three SWE-bench Verified runs, Astra develops systems centered on retrieval and context resets, bash interaction and history recovery, and execution feedback and context handling, achieving test scores of 19.76%, 28.85%, and 22.13%, respectively. All three runs retain the base model, yet their scores span 9.09 percentage points. These results illustrate the sensitivity of demanding agentic coding tasks to research strategy: different routes yield different outcomes for the same researcher and base model.

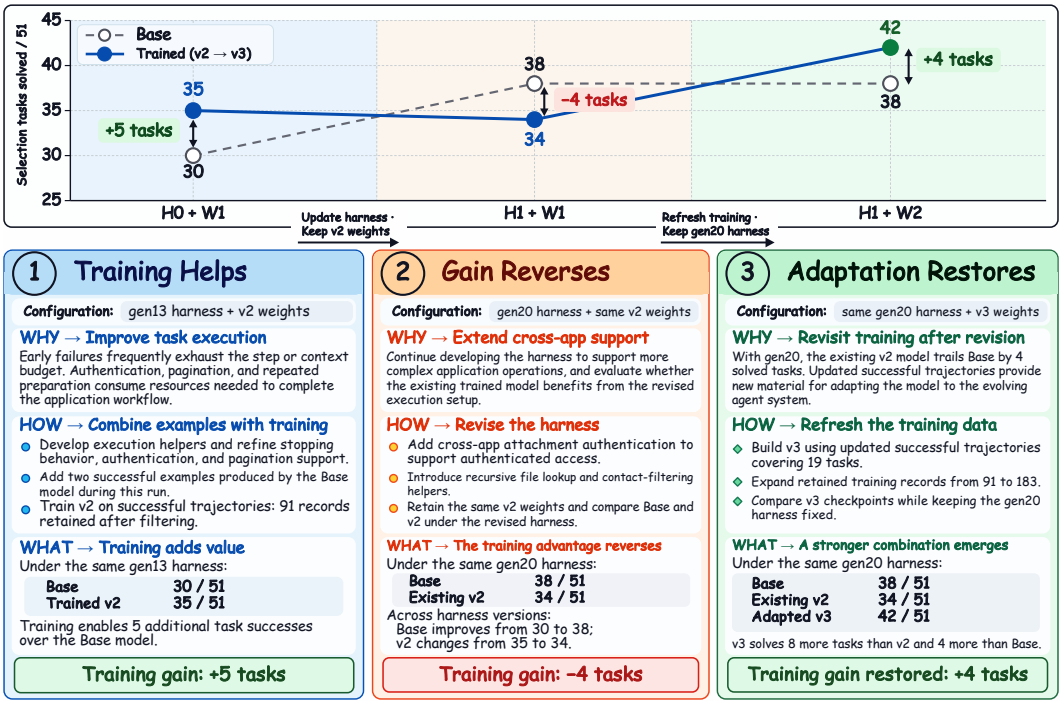

Figure 5: **A case study of the interaction between the harness and model weights.** In an Astra development run on AppWorld, training improves performance under the earlier harness, but its benefit reverses after a harness revision. Further training on refreshed data restores the gain under the revised harness. The top panel compares the number of tasks solved by the base and trained models on the same 51-task selection set; the bottom panels summarize the motivation, changes, and outcomes at each stage.

Specific interventions also yield substantial variation within the same broad strategy. All three Astra TB-Lite runs pair the base model with a direct-command-execution harness, optimizing output parsing, execution feedback, and recovery mechanisms. Their test scores are 17.50%, 31.03%, and 24.76%, spanning 13.53 percentage points. Together with the SWE-bench Verified results, this variation highlights the importance of both the development route and the implementation of individual interventions in agentic coding. By comparison, the main results on AppWorld and BFCL show clearer performance separation among researchers. Task structure thus shapes how research capability manifests: open-ended coding tasks expose run-to-run variation in strategy exploration, while structured interaction tasks reveal clearer differences among researchers.

6 Conclusion

We introduced **CoEvoBench**, a benchmark for evaluating LLM researchers on autonomous model–harness improvement across five agentic domains. Researchers choose and implement interventions using development feedback within a bounded resource budget, and deliver a model–harness pair for held-out evaluation. Our evaluation reveals substantial system improvements alongside challenges in turning successive feedback into further gains. Gains concentrate in key observed jumps, and the value of model updates changes with the harness. Together, these findings highlight the importance of adapting research interventions as the target model–harness system evolves. CoEvoBench evaluates these capabilities through final system performance and development trajectories; future work can extend its coverage to additional target models and examine how research performance scales with development resources.

AI USE STATEMENT

We used LLMs to assist with writing and language polishing, literature retrieval and discovery, and drafting parts of the manuscript. The authors take full responsibility for the final content, including AI-assisted text and claims.

REFERENCES

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, 2026. URL https://arxiv.org/abs/2507.19457.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering, 2025. URL https://arxiv.org/abs/2410.07095.

Mingju Chen, Can Lv, Guibin Zhang, Heng Chang, and Shiji Zhou. HarnessForge: Joint Harness and Policy Evolution for Adaptive Agent Systems, 2026a. URL https://arxiv.org/abs/2606.01779.

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, 2026b. URL https://arxiv.org/abs/2606.14249.

Wanyi Chen, Xiao Yang, Xu Yang, Tianming Sha, Qizheng Li, Zhuo Wang, Bowen Xian, Fang Kong, Weiqing Liu, and Jiang Bian. Agent^2 RL-Bench: Can LLM Agents Engineer Agentic RL Post-Training?, 2026c. URL https://arxiv.org/abs/2604.10547.

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

Yizhe Chi, Wenyi Li, Deyao Hong, Xiaoqiu Wang, Mingju Gao, Kaisen Yang, Bingxiang He, Youjie Zheng, Calvin Xiao, and Qinhuai Na. AI4AI-Bench: Benchmarking LLM Agents in Algorithmic Design for Recursive Self-Improvement, 2026. URL https://arxiv.org/abs/2608.20318.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench Verified. OpenAI, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Tianyi Guan, Yiding Wang, Haotong Yang, Siyuan Cao, Shirui Liu, Yi Hu, Jiaqi Li, and Muhan Zhang. ContinualSkillBench: Can LLM Agents Truly Evolve Their Capabilities?, 2026. URL https://arxiv.org/abs/2608.03874.

Zhiyuan He, Siwei Zhang, Zhiwen Zhou, Yuqing Yang, Yu Kang, Yuge Zhang, Luna K. Qiu, Tin Yan Tsui, Jiahang Xu, and Chong Luo. Agent Lightning v1.0: Towards Harnessed Agentic RL, 2026. URL https://arxiv.org/abs/2608.17528.

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

Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems, 2025. URL https://arxiv.org/abs/2408.08435.

Lisheng Huang, Chen Yang, Hao Zhou, Huatong Song, Zongchao Chen, Ran Le, Yang Song, Wayne Xin Zhao, and Tao Zhang. Evo-Bench: Can Language Models Improve Agent Harness?, 2026. URL https://arxiv.org/abs/2608.09096.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating Language Agents on Machine Learning Experimentation, 2024. URL https://arxiv.org/abs/2310.03302.

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

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery, 2024. URL https://arxiv.org/abs/2408.06292.

Huanzhi Mao, Amitoj Singh, Pracheeti Shikarkhane, Kai Wen, Shishir G. Patil, and Joseph E. Gonzalez. BFCL V4: Memory. Berkeley Function-Calling Leaderboard official release, July 2025. URL https://gorilla.cs.berkeley.edu/blogs/16_bfcl_v4_memory.html.

Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. RSIBench-Data: Benchmarking Data-Centric Research for Recursive Self-Improvement, 2026. URL https://arxiv.org/abs/2607.25886.

OpenThoughts-Agent Team, Snorkel AI, and Bespoke Labs. OpenThoughts-TBLite: A high-signal benchmark for iterating on terminal agents, 2026. URL https://www.open-thoughts.ai/blog/openthoughts-tblite.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training Software Engineering Agents and Verifiers with SWE-Gym, 2025. URL https://arxiv.org/abs/2412.21139.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. PostTrainBench: Can LLM Agents Automate LLM Post-Training?, 2026. URL https://arxiv.org/abs/2603.08640.

Quan Shi, Keshav Dhandhania, Karthik Narasimhan, and Victor Barres. τ^{τ}-Bench: An Environment for End-To-End, Realistic Agent Construction, 2026. URL https://arxiv.org/abs/2609.04611.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning, 2023. URL https://arxiv.org/abs/2303.11366.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's Ability to Replicate AI Research, 2025. URL https://arxiv.org/abs/2504.01848.

Zihan Tan, Leixin Sun, Zitong Shi, Yitao Liu, Jiajun Wu, Nathaniel Brooks, Jiaru Qian, Xiaoran Shang, Suyuan Huang, Yi Ding, Yangxu Liao, Mukai Li, Qiushi Sun, Shudong Liu, Xuankun Rong, Xiaohang Yu, Zhuo Chen, Hejia Geng, Chenxin Li, Aozhou Wang, Zengji Tu, Robert Tang, Yuxin Zhan, Eric Jiang, Yuxin Wu, Jianqing Zhang, Xiao Liang, Fang Wu, Haochi Zhang, Alexander Marlow, and Guancheng Wan. MetaRSI / RSI2: A Meta-Recursive Self-Improving System for Recursive Self-Improving Systems Themselves, 2026. URL https://arxiv.org/abs/2609.06396.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024. URL https://aclanthology.org/2024.acl-long.850/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models, 2023. URL https://arxiv.org/abs/2305.16291.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts, 2025. URL https://arxiv.org/abs/2411.15114.

Siwei Wu, Jincheng Ren, Yizhi Li, Haau-Sing Li, Chengran Yang, Yuxuan Zhang, Weicheng Gu, Jian Yang, Riza Batista-Navarro, Chuanyi Zhang, Xianglong Liu, Ming Zhou, Bryan Dai, and Chenghua Lin. ModularRSI: Modular and Generalizable Recursive Harness Self-Improvement, 2026. URL https://arxiv.org/abs/2609.14857.

Shuhan Xue, Zixin Ding, Yichen Shen, Yinjie Wang, Zhenfei Yin, Yingcheng Wu, Yuxin Chen, Mengdi Wang, and Ling Yang. PAST-Bench: Benchmarking the Foundations of Recursive Self-Improvement in Personal Agents, 2026a. URL https://arxiv.org/abs/2608.04003.

Shuhan Xue, Jianyuan Zhong, Ziyuan Nan, Wenbin Li, Zhaochen Yu, Jinchao Ding, Qiang Gao, Pengyu Zhan, Yuntong Zhang, Tian Cheng, Zhenfei Yin, Yingcheng Wu, and Ling Yang. ScienceBuddy: Recursive-in-Recursive Self-Improvement for Interactive Scientific Agents, 2026b. URL https://arxiv.org/abs/2609.17523.

Zhou Yu, Bin Bi, Shiva Kumar Pentyala, Shubham Mehrotra, Sougata Chaudhuri, Shilpa Bhagavath, Zeyuan Chen, Ran Xu, Phil Mui, James Zhu, and Sitaram Asur. Co-Evolving Harnesses and Models: On-Policy Correction Helps Weaker Models Catch Up Where Imitation Fails, 2026. URL https://arxiv.org/abs/2609.09134.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling Generalized Agent Abilities for LLMs, 2023. URL https://arxiv.org/abs/2310.12823.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents, 2026a. URL https://arxiv.org/abs/2505.22954.

Yifan Zhang, Yutong Dai, Juntao Tan, Luyu Yang, Rishi Mullur, Thai Hoang, Zhiyuan Hu, James Zhu, Phil Mui, Silvio Savarese, Ran Xu, and Zeyuan Chen. DarwinX: Evolving Agent Harnesses Through Natural Selection, 2026b. URL https://arxiv.org/abs/2608.07545.

A Researcher Workspace File Tree

Each development run uses an independent workspace containing task instructions, an entry point to the initial model, the editable harness, and development-evaluation tools. In the tree below, <benchmark> denotes the run's benchmark: swe, tb, appworld, tau, or bfcl.

```
workspace/
|-- TASK.md                              Task and interface instructions
|-- base                                 Initial model weights
|-- harness/<benchmark>/                 Editable harness
|-- tasks_visible/<benchmark>/           Visible task material
|-- anonymous_pool_<benchmark>.txt       Anonymous evaluation task IDs
|-- eval                                 Development evaluation entry
|-- timer.sh                             Research budget query
|-- model_path.txt                       Selected target model weights
`-- out/
    |-- ROUND.md                        Run-specific task and resources
    `-- <benchmark>_final/              Nominated final candidate
        |-- harness/                    Candidate harness
        |-- model_path.txt              Candidate weights path
        |-- meta.json                   Candidate metadata
        `-- serve_config.json           Model serving configuration
```

The researcher edits harness/, trains and selects model weights, and calls eval to compare candidates. Training scripts, training material, and derived weights are created in the workspace as research proceeds. The final candidate directory is created for an explicit nomination; candidate freezing follows Appendix E.

B Individual Experiment Results

Table 2 reports three independent development runs per researcher–benchmark pair.

Table 2: **Individual experiment results on the frozen Test partition (%).** Three runs and their mean are shown for each benchmark. All researcher models use the high configuration.

Benchmark	Run	Kimi K3	Qwen3.8-max	GLM-5.3	Claude Opus 5	GPT-6 Astra
SWE-bench Verified	Run 1	0.79	17.79	4.35	24.11	19.76
	Run 2	0.79	0.40	23.72	20.55	28.85
	Run 3	15.42	16.21	20.16	27.67	22.13
	Mean	**5.67**	**11.46**	**16.07**	**24.11**	**23.58**
TB-Lite	Run 1	6.00	20.03	23.47	14.17	17.50
	Run 2	15.03	18.15	16.00	20.68	31.03
	Run 3	3.25	17.62	18.47	24.17	24.76
	Mean	**8.09**	**18.60**	**19.31**	**19.67**	**24.43**
AppWorld easy	Run 1	5.88	7.84	5.88	37.25	37.25
	Run 2	1.96	0.00	13.73	49.02	43.14
	Run 3	1.96	7.84	17.65	23.53	33.33
	Mean	**3.27**	**5.23**	**12.42**	**36.60**	**37.91**
τ^2-bench Telecom	Run 1	17.78	11.67	18.33	16.11	37.22
	Run 2	2.78	18.33	20.00	17.78	96.11
	Run 3	22.78	10.00	20.00	74.44	88.33
	Mean	**14.44**	**13.33**	**19.44**	**36.11**	**73.89**
BFCL-V4	Run 1	13.56	4.17	2.00	2.83	26.10
	Run 2	13.96	5.06	4.00	19.75	19.54
	Run 3	7.38	4.29	2.83	20.88	31.98
	Mean	**11.63**	**4.51**	**2.94**	**14.49**	**25.88**

C BENCHMARK TASKS AND SCORING

CoEvoBench covers software repair, terminal operation, application interaction, telecom services, and function calling. Researchers improve target-agent performance by training the target model, modifying its harness, or combining both approaches.

C.1 TASK POOLS AND SPLIT CONSTRUCTION

We partition tasks into visible development (Visible), selection (Selection), and final testing (Test) sets, with sizes shown in Table 3. Fixed task manifests group or stratify tasks according to the structure of each domain.

Benchmark	Visible	Selection	Test	Total	Grouping / stratification
SWE-bench Verified	40	200	253	493	Repository
TB-Lite	20	40	40	100	Task difficulty
AppWorld easy	27	51	51	129	Scene / source split
τ^2-bench Telecom	23	46	45	114	Read-only / write
BFCL-V4	235	470	560	1,265	Task / memory groups

Table 3: **Frozen task partitions.** BFCL counts scoring records, including memory QA across backends. Each of the 45 Telecom Test tasks is evaluated four times. AppWorld groups by scene; BFCL groups by task family and memory scenario.

SWE-bench Verified. The software-repair pool contains 493 tasks. Given the diversity of repository-level repair tasks and their binary success criterion, we use 200 tasks in Selection to provide a finer-grained evaluation signal for comparing candidate models and harnesses during development. Visible provides task statements; gold patches and private tests are managed by the evaluator. The target agent edits the repository; the verifier checks whether the issue is resolved.

TB-Lite. The terminal-operation pool contains 100 tasks, stratified by difficulty. Visible, Selection, and Test contain 20, 40, and 40 tasks, respectively. The agent interacts with the task environment through a terminal harness and receives the evaluator's task reward, including partial credit.

AppWorld easy. We select 129 difficulty-1 tasks from the normal and challenge collections, covering 43 scenes with three variants per scene. Variants of the same scene are assigned together, yielding 9 Visible scenes, 17 Selection scenes, and 17 Test scenes. Assignment is stratified by task source. The agent generates Python code that calls application APIs; task goal completion (TGC) measures complete-task success.

τ^2-bench Telecom. We use the 114-task Telecom *base* subset, stratified by read-only and write operations. Each of the 45 Test tasks is evaluated four times, giving 180 trials. The platform controls the user simulator and task environment; success requires passing all task checks.

BFCL-V4 multi-turn and memory. The pool contains 800 multi-turn records, with 200 in each of four categories, and 465 memory QA records, with 155 for each of three backends. Multi-turn variants in the same task family are assigned together, yielding 160/320/320 records. Each memory scenario is grouped with its prerequisite conversations and QA records across backends, yielding 75/150/240 records. Each memory QA evaluation independently replays its prerequisite conversation; accuracy is calculated per QA record.

C.2 METRIC DEFINITIONS

All scores are displayed as percentages. For SWE-bench Verified and AppWorld, let $y_i \in \{0, 1\}$ indicate complete-task success; the score is $100 \sum_i y_i/N$. TB-Lite uses $100 \sum_i r_i/N$, where $r_i \in [0, 1]$ is the task reward.

For Telecom, let y_{ik} denote success on task i in trial k. We report mean single-trial success:

$$s_{\text{Telecom}} = \frac{100}{45 \times 4} \sum_{i=1}^{45} \sum_{k=1}^{4} y_{ik}. \tag{8}$$

For BFCL, let a_c be accuracy in multi-turn category c and m_d accuracy for memory backend d, each on the unit interval. The primary score is

$$s_{\text{BFCL}} = 100 \left(0.6 \, \frac{1}{4} \sum_{c=1}^{4} a_c + 0.4 \, \frac{1}{3} \sum_{d=1}^{3} m_d \right). \tag{9}$$

Test has equal record counts within each component, so the expression equals $100(0.6C_{\text{MT}}/320 + 0.4C_{\text{Mem}}/240)$, where C denotes the number of successful records in the corresponding component.

D Researcher and Target-Agent Configuration

D.1 Researcher Configurations and Resources

We use GPT-6 Astra, GLM-5.3, Qwen3.8-max, Kimi K3, and Claude Opus 5, all with the high configuration. Astra runs through Codex, while the other models run through Claude Code. Each researcher configuration comprises the model and its command-line tool.

Each development run receives a budget of 24 hours and four NVIDIA H20 GPUs, with early stopping permitted. Training, rollout generation, and model serving share these resources. The launcher sets the deadline, and the researcher can query the remaining budget.

D.2 Initial Target Agents

Target agents start from Qwen3.5-9B-Base. Researchers may retain the original weights or train derivatives using permitted data. Qwen3.5-9B-Instruct is used to construct the reference systems. Harness improvements include prompts, tool interaction, memory, and execution loops, relating to the harness optimization studied in Evo-Bench (Huang et al., 2026).

Software repair and terminal operation. SWE-bench Verified starts from the SWE-agent agent–computer interface, while TB-Lite starts from the terminal-oriented Terminus harness. Both connect to the evaluator through the `minisweagent` interface, exposing action parsing, tool interaction, observation formatting, context management, and the execution loop. The initial SWE-bench Verified configuration uses strict action parsing, truncated observations, and a context limit. The selected local model generates target-agent responses.

AppWorld. The initial harness provides a minimal code-generation protocol. Researchers may modify agent construction and next-code generation, while the application environment and scorer remain fixed. The reference uses ReAct with full demonstrations and solving instructions.

BFCL. The initial harness combines a fixed execution loop with editable prompting and history-selection policies. It retains one ordinary user turn and uses replacement-style memory notes. Researchers may modify turn initialization and message selection; tool execution, environment transitions, and scoring remain fixed.

Telecom. The initial harness uses text-based ReAct and generates strict JSON actions. Native tool-schema injection and automatic action repair are disabled. GPT-4o simulates users (temperature 0, 600-token output limit); GPT-4.1 judges natural-language assertions and environment interfaces.

Benchmark	Weak initial harness	Stronger reference harness
SWE-bench Verified	SWE-agent interface; strict parsing, truncated observations, and capped context.	mini-swe reference configuration.
TB-Lite	Terminal-oriented Terminus seed.	mini-swe reference configuration.
AppWorld	Minimal code-generation protocol, without full demonstrations or solving strategies.	ReAct with the reference demonstrations and solving prompt.
Telecom	Text ReAct with strict JSON actions; no native tool schema or automatic repair.	Telecom reference integration.
BFCL-V4	Fixed execution loop; one ordinary user turn and replacement-style memory notes.	BFCL reference integration and policy.

Table 4: **Initial and reference harnesses used in our experiments.** The initial and stronger reference systems use Qwen3.5-9B-Base and Qwen3.5-9B-Instruct, respectively.

D.3 Execution Limits and Serving Configuration

Table 5 lists the initial configurations used for each benchmark. Development evaluations also record the candidate weights, harness, and serving configuration.

Setting	SWE-bench Verified	TB-Lite	AppWorld	Telecom	BFCL
Context length	32,768	32,768	65,536	32,768	65,536
Output tokens / request	5,120	5,120	3,000	1,024	4,096
Steps / loop constant	80	120	50	200	20
Temperature	0	0	0	0	0.001 requested
Task wall limit (s)	900	600	None added	1,200	None added

Table 5: **Initial configurations used in our experiments.** BFCL uses `count > limit` loop semantics; serving raises the requested temperature of 0.001 to 0.01. "None added" means that the integration adds no task-level wall limit.

E Permitted Research and Researcher Instructions

E.1 Editable Components and Data Boundary

Researchers may modify weights, the harness, or both. Available operations include diagnosing Visible failures; adjusting prompts, parsers, memory, and context; constructing and filtering training data; generating rollouts; selecting training recipes; and comparing and revising candidates. Data construction and selection relate to data-centric research in RSIBench-Data (Meng et al., 2026).

Visible tasks support debugging, trajectory generation, and training. Network access is enabled, with datasets that directly expose benchmark task data blocked. Researchers may independently acquire permitted public or synthetic training material and record its sources and transformations. Selection/Test solutions, private tests, and per-task trajectories are protected: researchers may not retrieve them or construct data targeted at those instances.

The fixed `qwen3.8-27b` teacher may generate permitted training data within a $100 calling budget per development run; evaluation tasks are executed by the target student model. Research uses the supplied Base or its permitted derivatives within the assigned workspace and resources.

E.2 Development Feedback and Candidate Freezing

Researchers receive execution details on Visible and score feedback on Selection. Selection task statements and trajectories remain hidden. Development evaluations record task subsets and repetition counts; Test is reserved for final evaluation.

One model–harness pair is frozen at the end of development. The researcher nominates a candidate in `out/<benchmark>_final/`; without an explicit nomination, the submission is the workspace harness and the weights selected by `model_path.txt` at the deadline, defaulting to

Base. The operator records candidate identities, supplies replay metadata, and runs Test. Ordinary final evaluation leaves the frozen weights and harness unchanged; weight reversion required by the integrity policy follows Appendix I.

E.3 Common Task-Brief Excerpt

The common task instructions are excerpted below. Each domain additionally provides interface specifications, Visible material, evaluation commands, and resource information.

> Given the supplied 9B Base model and the initial harness for a benchmark, improve the target system within the assigned time and compute budget. You may modify the weights, the harness, or both, and choose whether to train.
>
> Use Visible tasks and permitted evaluation feedback to diagnose failures and compare candidates. You may acquire, generate, and transform permitted training material. Do not retrieve protected Selection/Test solutions, private tests, or trajectories, or construct data targeted at those instances.
>
> Use the supplied Base or its permitted derivatives. The fixed teacher is for training-data generation; the student independently completes evaluation tasks. Keep benchmark interfaces runnable and respect scoring, timing, and resource boundaries.
>
> Select and freeze a final model–harness pair for final evaluation by the operator. You may stop development early.

F Result Aggregation and Independent Runs

F.1 Baselines and Cross-Domain Aggregation

The Base reference combines the original weights with the initial harness. The stronger reference combines Qwen3.5-9B-Instruct with a reference harness: mini-swe for SWE-bench Verified and TB-Lite, ReAct for AppWorld, and the respective reference integrations for Telecom and BFCL.

For benchmark b, Base and stronger-reference scores B_b and I_b define fixed normalized weights:

$$w_b = \frac{(I_b - B_b)^{-1}}{\sum_j (I_j - B_j)^{-1}}, \qquad \text{Avg.} = \sum_b w_b s_b. \tag{10}$$

Here s_b is the researcher model's three-run mean on benchmark b. All researcher models use the same weights, shown in Table 6.

Benchmark	SWE-bench Verified	TB-Lite	AppWorld	Telecom	BFCL-V4
Weight (%)	25.18	16.52	21.80	7.39	29.11

Table 6: **Fixed scoring weights.** Displayed weights are rounded; calculations use their full normalized precision. These constants weight the benchmark means of the runs retained in Table 2.

G Research Runtime and Performance

Figure 6 compares average benchmark performance with average research time. Each point represents one researcher model using the high configuration.

Score and time calculation. The vertical coordinate is the weighted average Test score from Table 1, computed using the benchmark means in Table 2 and the fixed weights in Table 6. The horizontal coordinate is the arithmetic mean of available development durations, with equal weight per run. Timing spans the start to the end of research. Final Test evaluation and subsequent controls are timed separately.

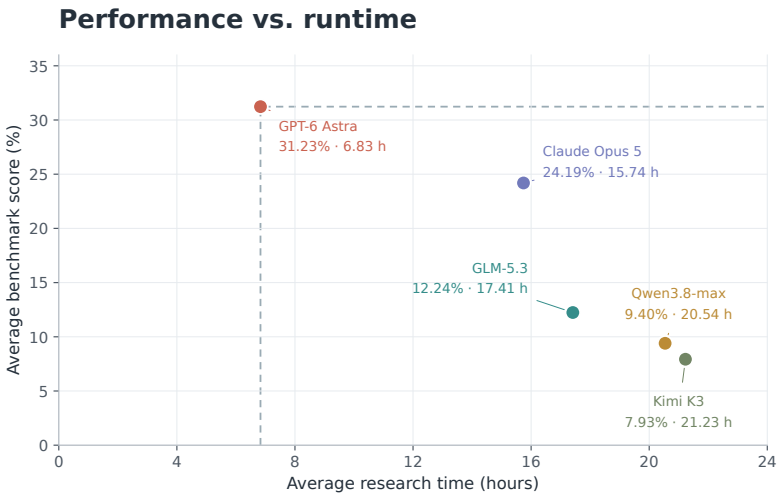

Figure 6: **Average performance versus average research time.** The horizontal axis gives average research hours; the vertical axis gives the weighted average Test score. The dashed line marks the efficiency frontier of the displayed means; upper-left points combine higher scores with less time.

Astra achieves 31.23% at an average of 6.83 hours; Opus, 24.19% at 15.74 hours; GLM, 12.24% at 17.41 hours; Qwen, 9.40% at 20.54 hours; and Kimi, 7.93% at 21.23 hours.

H Development Trajectories and Case Studies

H.1 Trajectory Records and Candidate Comparisons

Trajectories link candidate harnesses, model weights, training recipes, and evaluation feedback within each development run. The cases below correspond to specific runs in Table 2, using Selection scores to analyze development decisions and reporting the final candidates' Test scores.

H.2 AppWorld: Reversing and Restoring a Training Gain

In Astra AppWorld Run 2, the researcher first improves code continuation, API-documentation discovery, authentication, pagination, and context management. Two complete examples from successful Visible trajectories form the **example-enhanced harness**. Initial LoRA training increases Selection success from 30/51 to 35/51 under this harness.

The researcher adds attachment authentication, recursive live-path lookup, and relationship-based contact filtering. First attempts use temperature zero, while repeated failures retain an escape strategy. This **tool-enhanced harness** raises Base to 38/51, while the initial trained weights achieve 34/51. The paired results in Table 7 give a training-gain difference of $(34 - 38) - (35 - 30) = -9$ tasks.

Harness	Weights	Visible successes	Selection successes	Gain over same-harness Base
Example-enhanced	Base	16/27	30/51 (58.82%)	—
Example-enhanced	Initial training	17/27	35/51 (68.63%)	+5 tasks
Tool-enhanced	Base	17/27	38/51 (74.51%)	—
Tool-enhanced	Initial training	17/27	34/51 (66.67%)	−4 tasks
Tool-enhanced	Expanded training	20/27	42/51 (82.35%)	+4 tasks

Table 7: **Model–harness comparisons in Astra AppWorld Run 2.** Each Selection result evaluates the same 51 tasks once. Trained weights are first-epoch checkpoints; model comparisons within each harness version were checked against harness file fingerprints.

The researcher expands the successful-trajectory data and increases the maximum training sequence length from 16,384 to 32,768. The expanded training reaches 42/51 under the tool-enhanced harness, restoring a positive training gain. Table 8 gives the two training configurations.

Training setting	Initial training	Expanded training
Input / usable records	99 / 91	189 / 183
Maximum sequence length	16,384	32,768
LoRA rank / alpha	16 / 32	16 / 32
Learning rate	8×10^{-5}	8×10^{-5}
Gradient accumulation	4	4
Checkpoint compared	Epoch 1	Epoch 1

Table 8: **Training configurations for the AppWorld coupling case.** Records exceeding the sequence-length limit are filtered. Expanded-training inputs cover 19 tasks. Both configurations use seed 100 and train for two epochs; Table 7 compares first-epoch checkpoints.

The final candidate combines the tool-enhanced harness with the first-epoch checkpoint from expanded training. The operator converts the serving configuration to the evaluator's accepted format and adjusts the service port, preserving the nominated weights and harness. The candidate solves 22/51 Test tasks (43.14%). The paired Selection comparisons trace the training gain from $+5$ to -4 solved tasks after the harness change, then to $+4$ after jointly expanding the data and maximum training sequence length.

H.3 AppWorld: Harness Optimization and Self-Distillation

In Opus AppWorld Run 2, the researcher constructs a harness with initialization of time, user information, and credentials; login and pagination helpers; an API-name index; code-block stopping; empty-code and repeated-code resampling; and history compression. The fixed candidate is evaluated on Selection 35 times, yielding 1,159 successes out of 1,785 trials (64.93%).

The researcher uses 90 successful trajectories for self-distillation. Two first-epoch training variants achieve 56.90% (rank 8, learning rate 2×10^{-5}) and 47.70% (rank 16, learning rate 5×10^{-5}). Removing the pre-submission self-check then yields 741 successes across 22 Selection evaluations, with 1,122 trials and a mean of 66.04%. The final candidate uses this harness, which removes one generation step, with Base, achieving 49.02% on Test.

The complementary-gain example in the main text, which improves Test success from 0/51 to 10/51 through harness changes and then to 19/51 through training, corresponds to Opus AppWorld Run 1, with a final score of 37.25%.

H.4 Coding Tasks: Interface Adaptation and Training Regressions

In Astra TB-Lite Run 1, training lowers Selection reward from Base's 8.59% to 4.25% under the same harness with the stopping configuration repaired; a Base few-shot variant achieves 5.44%. Another harness with JSON recovery achieves 13.10% with Base and becomes the final candidate, scoring 17.50% on Test. This run also repairs checkpoint-serialization issues and ultimately uses Base with an improved harness.

GLM SWE-bench Verified Run 2 trains the model to adapt to an execution interface aligned with public successful trajectories. The trained checkpoint solves 44/200 Selection tasks, with a confirmation evaluation at 43/200, while Base solves 0/200 under the same interface. This case illustrates model adaptation to a specific execution interface.

Astra's three SWE-bench Verified Test scores are 19.76%, 28.85%, and 22.13%; its three TB-Lite scores are 17.50%, 31.03%, and 24.76%. All use Base with the harness developed in the corresponding run. The score ranges are 9.09 and 13.53 percentage points, respectively.

I Evaluation and Retesting Settings

For experiments exhibiting hacking behavior, retesting uses Base with the evolved harness: the model weights are reverted to Qwen3.5-9B-Base, the run's evolved harness is retained, and its performance is measured on Test.

J Model–Harness Updates Across Development Runs

We examine model and harness updates through comparisons of recorded system configurations. Table 9 combines final-system evaluations on AppWorld with development comparisons on SWE-bench Verified, TB-Lite, and AppWorld. Across the three AppWorld test comparisons, harness revisions improve the initial system, and updated weights provide further gains under the final harness. Development comparisons show both positive and negative training gains. In the TB-Lite run, averaging base and trained weights yields a higher selection score than either set of weights alone.

Table 9: **Model and harness configurations across development runs.** Scores are percentages. Panel (a) reports AppWorld test scores with one trial per task. Panel (b) reports development evaluations; n is the number of evaluated tasks. Run numbers follow Table 2. Recorded harness versions are shown; SWE-bench Verified and TB-Lite compare named development configurations.

(a) AppWorld Test: initial and final components

Researcher / Run	Initial system	Base + final harness	Updated + final harness
Opus / Run 1	0.00	19.61	37.25
Opus / Run 3	0.00	5.88	23.53
GLM / Run 2	0.00	7.84	13.73

(b) Development configurations

Benchmark / Run	Evaluation	Configuration	Score
SWE-bench Verified	Selection, $n = 40$	Base	2.50
Qwen / Run 1		SFT epoch2	12.50
TB-Lite	Selection, $n = 40$	Base	11.57
Qwen / Run 3		SFT-v3	9.08
		0.5 Base $+ 0.5$ SFT-v3	16.83
AppWorld	Visible, $n = 27$	Harness v2 + Base	11.11
GLM / Run 2		Harness v2 + SFT-r1	7.41
		Harness v4 + Base	7.41
		Harness v4 + SFT-r3	18.52

The dynamic model–harness interaction in Figure 5, including paired configurations and training settings, is detailed in Appendix H.2.