IMPROVEANYTASK: AN AUTONOMOUS POST-TRAINING HARNESS FOR ITERATIVE MODEL SELF-IMPROVEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Adapting general-purpose large language models to specific tasks requires substantial human effort in designing data and training strategies. Sustaining improvement is especially challenging because model updates change the error distribution, requiring strategies to be continually refined. We introduce **ImproveAnyTask**, an autonomous post-training harness that improves task performance under a limited compute budget. Drawing inspiration from gradient-based parameter optimization, the harness organizes adaptation into error attribution, update-direction selection, and executable model updates. It combines metric-level and case-level analysis to identify a focal problem, then investigates research-backed strategies and compares their reported gains and reproduction difficulty. The selected strategy is translated into training data and a training configuration, with small-scale execution checks preceding full post-training. Subsequent evaluation guides model selection and further adaptation, while validated strategies and scripts are retained for reuse. Across 11 tasks, ImproveAnyTask achieves mean gains of 18.29 and 11.97 percentage points on the Base and Instruct models, respectively, with a maximum gain of 41.96 points, under a 24-hour budget with resources equivalent to eight H20 GPUs.

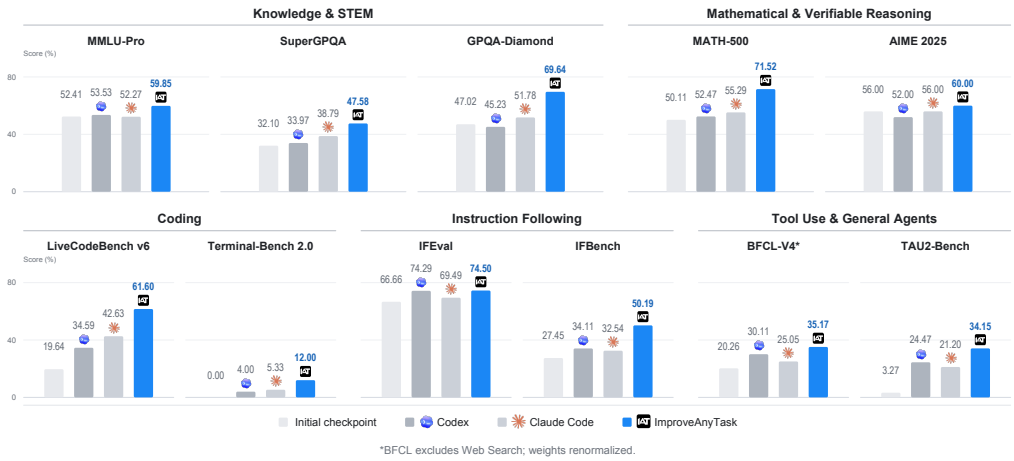


Figure 1: **Task-specific improvement of Qwen3.5-4B-Base.** We compare the initial checkpoint with models produced by Codex, Claude Code, and ImproveAnyTask across 11 tasks spanning five capability domains. Each run uses resources equivalent to eight H20 GPUs and stops after 24 hours or three optimization rounds, whichever is reached first. The mean gain is 18.29 percentage points, with a maximum of 41.96 points on LiveCodeBench v6. BFCL excludes Web Search and renormalizes the remaining weights.

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.

1 INTRODUCTION

Recent large language models, including GLM-5, Qwen3.5, and Kimi K3, have demonstrated strong capabilities across diverse tasks (GLM-5 Team, 2026; Qwen Team, 2026a; Kimi Team, 2026). However, strong general capabilities do not always translate into strong performance on specific tasks. Task-specific adaptation is therefore an important research direction for making these models more capable in real-world applications. Yet effective adaptation requires substantial effort in data curation, training strategy development, and model architecture design. Moreover, each model update can shift the error distribution, so sustained improvement requires continual refinement of the adaptation strategy.

Task-oriented post-training improves model performance through data selection, synthesis, distillation, and the design of supervised or reinforcement learning objectives (Ye et al., 2025; Guha et al., 2025; Hu et al., 2026; Ma et al., 2025). Researchers develop these interventions according to the target task and the model's weaknesses. Although many methods automate data construction or adapt training to the learner, selecting an appropriate strategy and revising it across successive checkpoints still require substantial human expertise.

Recent advances in autonomous research and model improvement have begun to automate literature investigation, implementation, and experimentation (Yamada et al., 2025; Tang et al., 2025; Nam et al., 2025). Other systems extend automation to post-training and iterative data curation (Rank et al., 2026; Du et al., 2026; Wang et al., 2026a). However, completing an experimental loop does not necessarily yield an effective improvement strategy. A promising method may address the wrong weakness, depend on unavailable resources, or fail during reproduction. Productive iteration therefore requires connecting the current model's errors with relevant research and reliable execution, while learning from previous outcomes. This motivates the following question:

Can an agent harness autonomously improve a general-purpose LLM on a target task under a limited compute budget?

To address this question, we propose **ImproveAnyTask**, an autonomous post-training harness that organizes task adaptation as an iterative model optimization process. Inspired by gradient-based parameter optimization, the harness connects error signals, update directions, and executable model updates. **Error Attribution** combines metric-level analysis with individual failure cases to identify a focal problem. **Update Direction** guides the agent to investigate multiple strategies and compare their reported gains, applicability, and reproduction difficulty. **Executable Model Updates** translates the selected strategy into training data and a training configuration, using small-scale execution and short training runs to identify implementation problems before full execution. Re-evaluation determines the outcome of each update and informs subsequent trials. Effective strategies and scripts are retained for reuse, while unsuccessful attempts provide evidence for revising future decisions. This process lets users adapt a model to a target task without manually redesigning data and training strategies at each iteration.

We evaluate 11 tasks spanning five capability domains using Qwen3.5-4B-Base and Qwen3.5-4B-Instruct under a 24-hour budget with resources equivalent to eight H20 GPUs. The Base-model gain averages 18.29 percentage points and reaches 41.96 points on LiveCodeBench v6 (Figure 1). The Instruct-model gain averages 11.97 points (Table 1).

Our main contributions are:

- **An autonomous post-training harness.** We introduce ImproveAnyTask to automate task-specific improvement of general-purpose LLMs under a limited compute budget, without requiring manually designed strategies for each trial.
- **An agent-guided model optimization process.** Inspired by gradient-based optimization, we organize model improvement into three modules: Error Attribution identifies what to improve, Update Direction selects how to improve it, and Executable Model Updates translates the strategy into data and training.
- **Cross-task evaluation.** Across 11 tasks under the same 24-hour budget, ImproveAnyTask achieves mean gains of 18.29 and 11.97 points on the Base and Instruct models, respectively.

2 Related Work

2.1 Task-Oriented Post-Training

Recent work improves task performance by shaping both training data and learning signals. **Data selection and curricula** emphasize example quality, diversity, and compatibility with the learner. Compact reasoning datasets can elicit existing capabilities, while student-aware and stage-aware curricula adapt which examples are used and when (Ye et al., 2025; Li et al., 2025; Huang et al., 2026; Hu et al., 2026). **Synthesis and distillation** expand supervision beyond existing labeled corpora: open data recipes scale reasoning and coding demonstrations (Guha et al., 2025; Ahmad et al., 2025), iterative prompt synthesis improves the question supply (Zhao et al., 2025), and failure-driven methods target missing knowledge with document-grounded supervision (Xu et al., 2026). **Task-structured supervision** makes success criteria explicit through rubric-guided reasoning (Yu et al., 2026a), executable software-repair tasks (Yang et al., 2025), and diverse function-calling examples (Greenstein et al., 2026). Complementarily, **training objectives and composition** determine how supervision translates into updates, including general-domain verification (Ma et al., 2025), conditional-expectation rewards (Xiao et al., 2026), and multi-domain training or model merging (Wang et al., 2026b). These advances provide powerful optimization ingredients, and several already automate data construction or adapt it to the learner.

However, selecting and revising these methods still requires **substantial human effort and time**. As training corrects some errors and introduces others, a previously effective strategy may no longer address the model's main weaknesses. Automated data construction alone does not resolve this need for **strategy adaptation as errors change**. ImproveAnyTask automates this repeated diagnosis and strategy revision, reducing the need for manual iteration.

2.2 Autonomous Research and Model Improvement

LLM agents increasingly automate empirical research rather than merely propose ideas. AI Scientist-v2 (Yamada et al., 2025) and AI-Researcher (Tang et al., 2025) connect literature analysis, implementation, and experimentation, while MLE-STAR (Nam et al., 2025) improves machine-learning pipelines through search and targeted refinement. For model adaptation, PostTrain-Bench (Rank et al., 2026) evaluates autonomous post-training under bounded compute. FT-Agent (Li et al., 2026) and AutoTrainess (Yu et al., 2026b) combine iterative planning, error feedback, execution checks, and trial records. LaMDAgent (Yano et al., 2025) searches end-to-end post-training pipelines, while TREX (Ma et al., 2026) combines literature research and execution with tree-based search and history reuse. LLMZero (Fang et al., 2026) and Pioneer Agent (Atreja et al., 2026) further explore training trajectories and deployment-time interventions. These studies establish iterative, end-to-end post-training as a shared setting.

A complementary line makes training data an explicit optimization variable. DataMaster (Du et al., 2026) searches external data and executable transformations, ANDES (Zhao et al., 2026) provides adaptive instruction synthesis, and CurateEvo (Wang et al., 2026a) evolves curation programs using post-training feedback. Curation-Bench (Kang et al., 2026) identifies an execution–research gap; its paper-adaptation scaffold requires agents to select prior methods, justify their applicability, and adapt them to available data and budgets. Related methods include SEAL (Zweiger et al., 2025), which learns self-updates, and AgentEvolver (Zhai et al., 2025) and CoEvolve (Yang et al., 2026), which evolve training tasks alongside the model. These approaches primarily optimize data strategies, with less emphasis on comparing candidate methods by expected benefit and reproduction difficulty. What is Missing from AI Post-Training AI (Lim et al., 2026) further finds that the evaluated post-training agents can repair execution problems but rarely reconsider their chosen strategies.

Beyond completing a training pipeline, sustained improvement requires **selecting effective interventions as model errors change** and **implementing them reliably within the budget**. ImproveAnyTask connects these decisions in a complete optimization loop: error attribution identifies one priority, research compares methods for that priority, and execution checks help implement the selected method. Re-evaluation then guides the next round. We study this integrated process across 11 tasks and two model initializations under a fixed budget, obtaining substantial gains over the evaluated general-purpose agent baselines.

3 METHODS

3.1 PROBLEM FORMULATION AND OVERVIEW

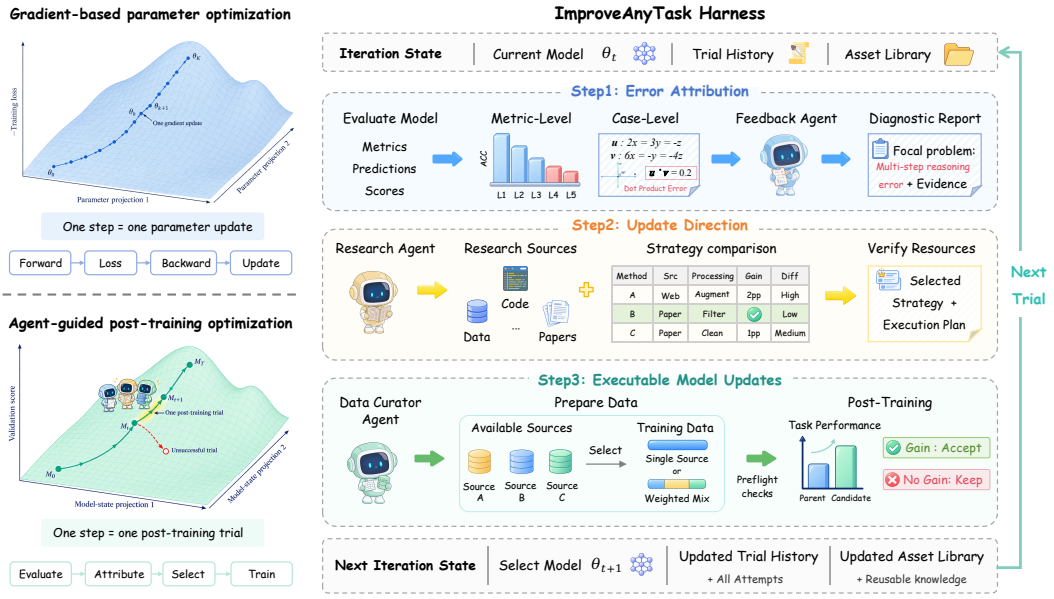


Figure 2: Overview of ImproveAnyTask. The harness treats task adaptation as iterative optimization: evaluation and dual-level error attribution identify a focal problem; research-grounded strategy exploration selects an update direction; data preparation and post-training produce a candidate model; and re-evaluation determines the next checkpoint and records reusable assets.

Problem Formulation. Given an initial model θ_0 and a target task B, our objective is to autonomously select and execute post-training strategies that improve its task performance within a time budget $\mathcal{B}$ of 24 hours, using resources equivalent to eight H20 GPUs:

$$\max_{\{s_t\}} J_B(\theta_{\text{final}}) \quad \text{s.t.} \quad T_{\text{total}} \leq \mathcal{B}. \tag{1}$$

Here, J_B denotes the primary task metric, evaluated on the validation split during optimization; s_t is the strategy adopted in trial t, and θ_{final} is the selected post-trained candidate. T_{total} measures end-to-end elapsed time; Section 4.1 specifies the timing and stopping rules.

Harness-Guided Model Optimization. As shown in Figure 2, ImproveAnyTask draws inspiration from gradient-based parameter optimization. Specifically, the harness first organizes evaluation of the current model θ_t on the validation split of task B, analogous to a *forward pass*. The **Error Attribution module** then jointly analyzes metrics and failure cases to extract *update-relevant signals*, analogous in role to *backpropagation*. It produces a diagnostic report F_t that identifies a focal problem p_t and its supporting evidence. The **Update Direction module** uses research evidence and prior experience to select a strategy s_t, providing an *update direction* for the current model. The **Executable Model Updates module** translates s_t into training data D_t and a training configuration h_t, then performs post-training to realize the *parameter update*, producing a candidate model $\tilde{\theta}_{t+1}$. The candidate is subsequently re-evaluated. This repeated evaluation–research–update loop enables the model to self-improve within the harness. Strategies and scripts validated as effective are retained for future trials.

3.2 ERROR ATTRIBUTION

The harness organizes an **error attribution module** to turn evaluation evidence into a prioritized optimization target for the current model θ_t. Analogous in role to *backpropagation*, this module extracts update-relevant signals from observed errors. Specifically, our harness first evaluates θ_t

on the validation split of task B, collecting aggregate metrics, individual predictions, and scoring results. It guides the agent to prioritize problems through **Dual-Level Error Attribution** and record the focal problem p_t and its evidence in a diagnostic report F_t for subsequent optimization.

Dual-Level Error Attribution. To produce actionable diagnoses, we examine errors at both the metric and case levels. Metric-level analysis identifies weaknesses in overall performance and available subtask metrics. Case-level analysis examines individual predictions and scoring evidence to identify recurring failure patterns. Together, they establish the scope of a weakness and explain how the model fails. Section 4.4 evaluates error attribution and focal-problem selection together.

Combining both sources of evidence, the harness guides the agent to select the focal problem from the candidate set $\mathcal{P}_t$:

$$p_t = \arg\max_{p \in \mathcal{P}_t} I_B(p), \tag{2}$$

where $I_B(p)$ denotes the assessed impact of problem p on the primary metric, based on metric and case evidence. It represents a relative priority, not a numerical prediction of post-training gains.

Subsequent research and execution focus on p_t, rather than all identified problems. Inspired by regularization, this restriction narrows the set of problems targeted in each trial. We observed that presenting multiple problems can lead the agent to favor easier secondary issues over the most consequential one.

3.3 Update Direction

To determine an *update direction* for θ_t, we design an **Update Direction module** that guides the agent to investigate methods relevant to the focal problem p_t. Unlike gradient-based optimization, our outer loop has no directly usable gradient over data strategies. The harness instead draws on evidence from papers, code, and datasets to select a strategy s_t and develop its execution plan.

Structured Strategy Exploration. The harness requires the agent to investigate multiple candidates for the same focal problem p_t before selecting a strategy. Each candidate is recorded in a shared format covering its source, applicability, reported gains, and reproduction steps. The module also assesses reproduction difficulty on a three-level scale. The lowest level corresponds to direct use of released training-ready data; the intermediate level requires additional processing, generation, or distillation; the highest level involves complex procedures or incomplete resources. These records support comparison across candidates before resources are committed to implementation.

The module then guides the agent to select strategy s_t through a qualitative comparison of applicability, reported gains, and reproduction difficulty. Reported gains provide evidence from prior settings, while reproduction difficulty helps assess whether the required implementation is practical under the available budget. Before execution, the agent verifies the required resources and key reproduction steps to ensure that the selected strategy can be implemented.

Outcome-Aware Reuse. As θ_t changes, its errors and the applicability of previous strategies may also change. Each round therefore includes research for the current model state, while reusing existing records to avoid repeated work. The candidate pool combines three sources:

$$\mathcal{S}_t = \mathcal{S}_t^{\text{new}} \cup \mathcal{S}_t^{\text{history}} \cup \mathcal{S}_t^{\text{library}}. \tag{3}$$

Here, $\mathcal{S}_t^{\text{history}}$ retains previously unselected candidates and the outcomes of executed trials, while $\mathcal{S}_t^{\text{library}}$ provides validated, reusable knowledge. These records help the agent revisit promising methods and avoid repeated failures. They inform strategy selection but do not replace validation on the current model.

3.4 Executable Model Updates

To realize the selected strategy s_t as a *parameter update*, the **Executable Model Updates module** guides the agent to acquire resources and implement processing tools to produce training data D_t and a configuration h_t. Training settings follow the selected method, with batch size and parallelism adapted to the current model θ_t and the available resources. The harness then performs post-training:

$$\tilde{\theta}_{t+1} = \mathcal{U}(\theta_t; D_t, h_t), \tag{4}$$

where $\mathcal{U}$ denotes the training procedure and $\tilde{\theta}_{t+1}$ is the resulting candidate model.

Execution-Guided Refinement. The harness organizes standard execution checks to assess whether the prepared data and training configuration implement s_t as intended. Before full execution, the agent applies the data preparation procedure to a small set of real samples. It checks whether their content, format, and training template match the requirements of s_t. A short training run verifies that the prepared samples can be loaded and parameter updates completed. When execution reveals errors or mismatches, the agent revises the processing tools, data pipeline, or h_t, then repeats the relevant checks. Full data preparation and training begin after these checks pass.

Evaluation and Reuse. The candidate $\tilde{\theta}_{t+1}$ is re-evaluated on the same validation split and under the same conditions as θ_t. Using F_t and the executed strategy s_t, the agent analyzes changes in the primary metric, whether the focal problem p_t has improved, and any new errors. These validation results guide subsequent model and strategy selection. If the candidate does not improve the validation score, the parent remains the starting point for subsequent trials.

Experiment records link every attempt, including failures, to its strategy, data, training settings, and results; effective strategies, scripts, and data are retained in the asset libraries for reuse.

4 Experiments

4.1 Experimental Setup

Models. We evaluate ImproveAnyTask on **Qwen3.5-4B-Base** and **Qwen3.5-4B-Instruct** (Qwen Team, 2026b;a), testing task adaptation from pretrained and post-trained initializations. Each run starts independently from the original checkpoint.

Tasks. Following the broad evaluation coverage used in recent frontier-model reports (Qwen Team, 2026a; Kimi Team, 2026; GLM-5 Team, 2026), we evaluate task-specific self-improvement on 11 tasks across five capability domains: (1) **Knowledge and STEM**: MMLU-Pro, SuperGPQA, and GPQA-Diamond (Wang et al., 2024; M-A-P Team, 2025; Rein et al., 2023); (2) **Mathematical and Verifiable Reasoning**: MATH-500 and AIME 2025 (Lightman et al., 2023; Mathematical Association of America, 2025); (3) **Instruction Following**: IFEval and IFBench (Zhou et al., 2023; Pyatkin et al., 2025); (4) **Coding**: LiveCodeBench v6 and Terminal-Bench 2.0 (Jain et al., 2024; Merrill et al., 2026); and (5) **Tool Use and General Agents**: BFCL-V4 and TAU2-Bench (Patil et al., 2025; Barres et al., 2025). For BFCL-V4, we exclude the web-search subset because it requires an external paid search service.

Compute Budget. Each run uses resources equivalent to **eight NVIDIA H20 GPUs** and stops after **24 hours** or **three optimization rounds**, whichever comes first. Timing spans the initial to final evaluation, including research, data preparation, validation, training, and failed attempts; unfinished work is interrupted at the deadline. Section 4.3 extends these limits.

Evaluation Protocol. Following FT-Agent (Li et al., 2026), all methods share a fixed split of approximately 15% validation and 85% held-out test data (Appendix A.1). Only validation data inform error analysis, strategy selection, and checkpoint selection; test data remain inaccessible during optimization. Within each run, we select the post-trained candidate with the highest validation score among those validated before the deadline; the initial checkpoint is excluded and reported separately as *Original*.

All experiments use the same official evaluation code, a 128K context window, non-thinking generation, and greedy decoding at temperature 0; empty or failed predictions receive zero credit. To reduce sampling variability during optimization, we also set the agent's API temperature to 0. Main and ablation results are averaged over two independent complete optimization runs per setting, each subject to the same budget. We log exact matches to evaluation queries in training data. Only BFCL aggregation changes, excluding Web Search and renormalizing weights (Appendix A.4).

Baselines. We compare against **Codex** (OpenAI, 2025) with GPT-5.6-Sol and **Claude Code** (Anthropic, 2025) with Opus 4.8. To our knowledge, no related specialized system provides a complete open-source implementation for our full 11-task suite. ImproveAnyTask also uses Opus 4.8. All systems share tool permissions, teacher-model access, and resource limits. Under **Direct Agent**, each baseline receives only the task, initial model, budget, and evaluation interface, then indepen-

Table 1: Mean results over two independent optimization runs across 11 tasks. Each run selects its post-trained candidate by validation score, excluding Original. BFCL uses the renormalized no-Web metric. All values are percentages.

	Qwen3.5-4B-Base				Qwen3.5-4B-Instruct			
Benchmark	**Original**	**Codex**	**Claude Code**	**Ours**	**Original**	**Codex**	**Claude Code**	**Ours**
Knowledge & STEM								
MMLU-Pro	52.41	53.53 (↑1.12)	52.27 (↓0.13)	**59.85** (↑7.44)	58.66	53.41 (↓5.25)	57.27 (↓1.38)	**69.33** (↑10.66)
SuperGPQA	32.10	33.97 (↑1.86)	38.79 (↑6.68)	**47.58** (↑15.48)	**49.78**	49.54 (↓0.23)	47.04 (↓2.73)	47.77 (↓2.00)
GPQA-Diamond	47.02	45.23 (↓1.78)	51.78 (↑4.76)	**69.64** (↑22.61)	75.59	73.21 (↓2.38)	78.57 (↑2.97)	**80.35** (↑4.76)
Mathematical & Verifiable Reasoning								
MATH-500	50.11	52.47 (↑2.35)	55.29 (↑5.17)	**71.52** (↑21.41)	51.29	56.00 (↑4.70)	57.41 (↑6.11)	**63.52** (↑12.23)
AIME 2025	56.00	52.00 (↓4.00)	56.00 (0.00)	**60.00** (↑4.00)	40.00	44.00 (↑4.00)	56.00 (↑16.00)	**64.00** (↑24.00)
Instruction Following								
IFEval	66.66	74.29 (↑7.62)	69.49 (↑2.83)	**74.50** (↑7.84)	82.78	83.44 (↑0.65)	80.61 (↓2.17)	**83.66** (↑0.87)
IFBench	27.45	34.11 (↑6.66)	32.54 (↑5.09)	**50.19** (↑22.74)	32.94	35.68 (↑2.74)	36.07 (↑3.13)	**54.50** (↑21.56)
Coding								
LiveCodeBench v6	19.64	34.59 (↑14.95)	42.63 (↑22.99)	**61.60** (↑41.96)	11.49	24.77 (↑13.28)	22.43 (↑10.93)	**40.62** (↑29.12)
Terminal-Bench 2.0	0.00	4.00 (↑4.00)	5.33 (↑5.33)	**12.00** (↑12.00)	10.66	13.33 (↑2.66)	14.66 (↑4.00)	**21.33** (↑10.66)
Tool Use & General Agents								
BFCL-V4[†]	20.26	30.11 (↑9.85)	25.05 (↑4.78)	**35.17** (↑14.91)	66.73	70.76 (↑4.02)	69.60 (↑2.86)	**78.33** (↑11.60)
TAU2-Bench	3.27	24.47 (↑21.20)	21.20 (↑17.93)	**34.15** (↑30.87)	36.68	39.21 (↑2.52)	40.25 (↑3.57)	**44.94** (↑8.25)
Overall Average[‡]	34.08	39.89 (↑5.80)	40.94 (↑6.85)	**52.38** (↑18.29)	46.96	49.39 (↑2.43)	50.90 (↑3.93)	**58.94** (↑11.97)

[†] BFCL weights: Non-Live, Live, and Irrelevance, 12.5% each; Multi-Turn, 37.5%; Memory, 25% (Appendix A.4).
[‡] Columns average all 11 tasks; gains are relative to Original. Bold marks the best result per model block; green/red denotes gains/losses.

dently researches methods, prepares data, implements tools, and trains the model. Codex provides a complete-system comparison, not an isolated test of harness design.

4.2 Main Results

Overall Results. Table 1 reports the main results. ImproveAnyTask achieves mean gains of 18.29 percentage points on Qwen3.5-4B-Base and 11.97 points on Qwen3.5-4B-Instruct. It exceeds Codex and Claude Code by 12.49 and 11.43 points on Base, and by 9.54 and 8.04 points on Instruct. These averages include the renormalized BFCL no-Web metric.

Task-Level Results. ImproveAnyTask improves all eleven Base-model tasks and ten Instruct-model tasks. The largest Base-model gain occurs on LiveCodeBench v6, from 19.64% to 61.60% $(+41.96)$, while the smallest occurs on AIME 2025 $(+4.00)$. For Instruct, the largest gains are on LiveCodeBench v6 $(+29.12)$, AIME 2025 $(+24.00)$, and IFBench $(+21.56)$. Notably, under our evaluation settings, the optimized Base model outperforms the optimized Instruct model on MATH-500 and LiveCodeBench v6.

On IFEval, ImproveAnyTask gains 7.84 and 0.87 percentage points on Base and Instruct, respectively. Section 4.3 examines the Base-model trajectory within and beyond the first 24 hours.

SuperGPQA-Instruct is the only task with a decrease, from 49.78% to 47.77% (-2.00), with changes computed before truncation. The recorded run contains three unsuccessful attempts. The first used 70K teacher-distilled examples, but only 12,747 were multiple-choice questions and the data were concentrated in physics, mathematics, and chemistry. The second used 15K student-consistent examples, of which 89.1% reproduced responses the student already answered correctly. The third increased the correction ratio to 80% using 1K targeted examples. These records suggest distribution mismatch and forgetting as possible explanations, but do not establish a single cause.

4.3 Scaling the Optimization Budget

We extend the IFEval Base experiment from 24 to 100 hours under the same resource allocation, timing rules, and generation settings as the main experiments. Removing the three-round limit allows eight candidate checkpoints. Figure 3 reports strict prompt-level accuracy; Appendix A.2 provides execution details.

The first two checkpoints use persona-conditioned instruction tuning and rubric-guided response selection, increasing accuracy from 66.66% to 74.50% within 24 hours. Adding a third source of general instruction data yields a smaller gain, reaching 74.72% at C3.

Targeted, execution-verified tuning reduces accuracy at C4. C5 restarts from C3 and adds targeted examples without replacing the broader mixture; C6 targets residual errors. C7 introduces constraint reasoning and self-verification in the training data, reaching 85.18%, 10.67 points above C2. A narrower continuation focused on precision constraints at C8 falls to 83.00%. Longer exploration finds a better strategy in this run, but does not guarantee improvement at every checkpoint.

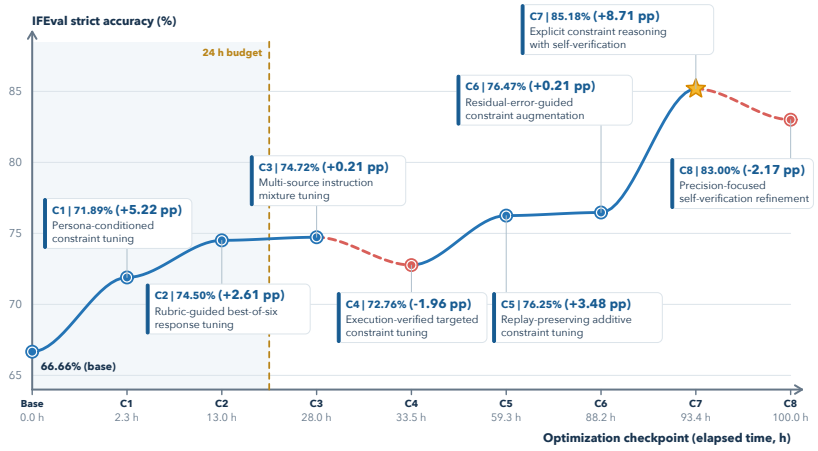


Figure 3: IFEval optimization over 100 hours. The dashed line marks 24 hours. Checkpoints are equally spaced; changes compare adjacent displayed checkpoints. C5 resumes from C3.

4.4 Ablation Study

Experimental Design. We evaluate three modules of ImproveAnyTask through module-level replacement experiments: (1) **Error Attribution**, including dual-level error analysis and focal-problem selection; (2) **Update Direction**, including structured strategy comparison, resource verification, reproduction-difficulty assessment, and outcome-aware reuse; and (3) **Execution-Guided Refinement** within **Executable Model Updates**, including sample-level data and template checks, short training runs, and iterative correction of data and training pipelines.

Within Error Attribution, metric- and case-level analysis provides evidence for prioritizing problems, while focal-problem selection turns that evidence into the next research target. We replace these procedures together to evaluate the complete process from diagnosis to problem selection.

Rather than removing an entire stage and breaking the optimization loop, each ablation replaces the corresponding mechanism with a basic alternative. *w/o EA* retains evaluation feedback but replaces structured dual-level attribution and explicit focal-problem selection with an unstructured summary. *w/o UD* retains ordinary research but removes the required comparison of candidates for the same focal problem, reproduction-difficulty assessment, resource verification, and outcome-aware reuse. *w/o ER* retains Executable Model Updates but removes our structured pre-execution checks and refinement procedure. The underlying agent remains able to inspect artifacts and repair errors in all variants. These replacements remove the harness-prescribed procedures, not the agent's general ability to analyze feedback, research methods, or debug code. Appendix A.3 specifies these replacements and their recorded execution paths.

Table 2: Ablation results for Qwen3.5-4B-Instruct, averaged over two independent optimization runs. Each component is replaced with a basic alternative while preserving the complete optimization loop. All values are percentages.

Benchmark	Original	w/o EA	w/o UD	w/o ER	Full Harness
MMLU-Pro	58.66	61.20 (↑2.53)	60.30 (↑1.63)	52.51 (↓6.15)	**69.33** (↑10.66)
MATH-500	51.29	56.00 (↑4.70)	56.70 (↑5.41)	54.11 (↑2.82)	**63.52** (↑12.23)
LiveCodeBench v6	11.49	24.77 (↑13.28)	33.03 (↑21.54)	28.57 (↑17.07)	**40.62** (↑29.12)
TAU2-Bench	36.68	39.21 (↑2.52)	41.36 (↑4.68)	38.09 (↑1.41)	**44.94** (↑8.25)
IFEval	82.78	82.35 (↓0.43)	83.22 (↑0.43)	80.17 (↓2.61)	**83.66** (↑0.87)
Overall Average	48.18	52.70 (↑4.52)	54.92 (↑6.74)	50.69 (↑2.50)	**60.41** (↑12.23)

Overall Average covers five benchmarks. Green/red values are changes from Original; bold marks the best result per row.

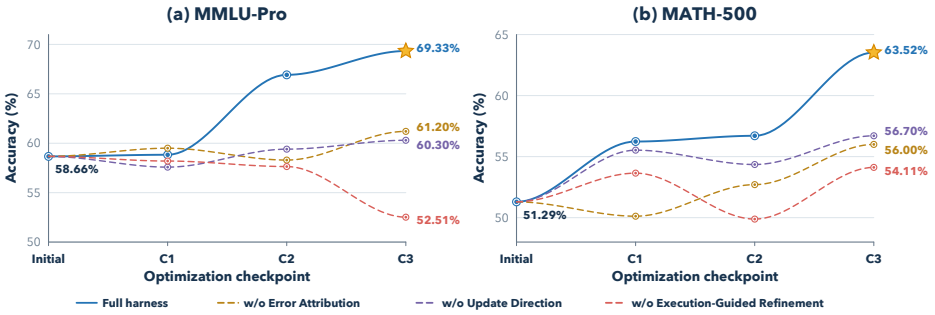


Figure 4: Optimization trajectories for Qwen3.5-4B-Instruct on MMLU-Pro and MATH-500. Initial and final scores match Table 2. Curves connect measured checkpoints for visualization.

We evaluate Qwen3.5-4B-Instruct on five representative benchmarks, one from each capability domain. We use the Instruct model to examine improvement beyond an already post-trained initialization. All runs follow the main experimental settings in Section 4.1.

The variants can select different training data and configurations as their optimization paths diverge. The comparisons therefore measure each module's overall effect under the same budget, not the isolated contribution of a single check or data choice.

Overall Effects. Table 2 reports a full-harness average of 60.41%, a gain of 12.23 percentage points over the initial model. Replacing Error Attribution, Update Direction, or Execution-Guided Refinement reduces the final average by 7.70, 5.49, and 9.72 points, respectively. Appendix A.3 describes the recorded strategies and execution issues.

Optimization Trajectories. Figure 4 illustrates the MMLU-Pro and MATH-500 trajectories. The full harness improves across successive checkpoints, whereas the replacement variants fluctuate and finish below it. Without Execution-Guided Refinement, MMLU-Pro ends below its initial score, while MATH-500 partially recovers after a decline.

5 Conclusion

We introduced ImproveAnyTask, an autonomous post-training harness that supports iterative model self-improvement through error attribution, research-grounded update selection, and executable model updates. Across 11 tasks, it achieves mean gains of 18.29 and 11.97 percentage points on Base and Instruct models under a 24-hour budget with resources equivalent to eight H20 GPUs. By automating repeated diagnosis, research, data preparation, and training, the harness reduces the need for manual trial design and intervention. This offers a practical way to adapt models for specific applications with less human effort and a fixed resource budget.

AI USE STATEMENT

This work studies an autonomous post-training harness built upon generative models. Generative models are therefore used as research, data-curation, and execution agents within the proposed method. AI tools also assisted manuscript editing and figure preparation. The authors take full responsibility for the content and conclusions of this work.

ETHICS STATEMENT

ImproveAnyTask uses released resources and model-generated data for task-specific post-training. These sources may contain biases, factual errors, or sensitive information, and higher task scores do not establish safety or fairness. Validation feedback is isolated from final testing. Exact-overlap auditing does not rule out semantic contamination in external data, and repeated use of validation feedback can still lead to validation-set overfitting. Autonomous code execution and external API calls also introduce privacy, security, and resource-use risks. Practical deployment therefore requires appropriate access controls and review of generated code and data. Third-party models, code, and datasets remain subject to their licenses and usage conditions.

REPRODUCIBILITY STATEMENT

Section 3 describes the optimization workflow, and Section 4.1 specifies the models, tasks, evaluation protocol, and resource limits. The user-facing task prompt is, for example, "Autonomously optimize the model for MMLU-Pro." This request initiates the harness workflow described in Section 3. Appendix A.2 provides checkpoint ancestry, data construction, training-set sizes, and training and generation settings for the extended IFEval run. Appendix A.3 describes the baseline replacements used in the ablations and the recorded strategies and execution issues for each task. We will release the complete source code and execution traces upon acceptance to support reproduction. Although we set the optimization agent's API temperature to 0, external resource availability, API behavior, and training randomness can still affect the full search trajectory.

REFERENCES

Wasi Uddin Ahmad, Sean Narenthiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. OpenCodeReasoning: Advancing Data Distillation for Competitive Coding, 2025. URL https://arxiv.org/abs/2504.01943.

Anthropic. Claude 3.7 Sonnet and Claude Code. Anthropic, 2025. URL https://www.anthropic.com/news/claude-3-7-sonnet.

Dhruv Atreja, Julia White, Nikhil Nayak, Kelton Zhang, Henrijs Princis, George Hurn-Maloney, Ash Lewis, and Urchade Zaratiana. Pioneer Agent: Continual Improvement of Small Language Models in Production, 2026. URL https://arxiv.org/abs/2604.09791.

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.

Yaxin Du, Xiyuan Yang, Zhifan Zhou, Wanxu Liu, Zixing Lei, Zimeng Chen, Fenyi Liu, Haotian Wu, Yuzhu Cai, Zexi Liu, Xinyu Zhu, WenHao Wang, Linfeng Zhang, Chen Qian, and Siheng Chen. DataMaster: Data-Centric Autonomous AI Research, 2026. URL https://arxiv.org/abs/2605.10906.

Haoyang Fang, Wei Zhu, Boran Han, Alex Zhang, Zhenyu Pan, Shuo Yang, Shuai Zhang, Jiading Gai, Peng Tang, Cuixiong Hu, Xuan Zhu, Huzefa Rangwala, George Karypis, and Bernie Wang. LLMZero: Discovering Adaptive Training Strategies for RL Post-Training via LLM Agents, 2026. URL https://arxiv.org/abs/2606.18388.

GLM-5 Team. GLM-5: From Vibe Coding to Agentic Engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Dan Greenstein, Zohar Karnin, Chen Amiraz, and Oren Somekh. Linguistic and Argument Diversity in Synthetic Data for Function-Calling Agents, 2026. URL https://arxiv.org/abs/2601.17829.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik Sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah Pratt, Vivek Ramanujan, Jon Saad-Falcon, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A. Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alexandros G. Dimakis, and Ludwig Schmidt. OpenThoughts: Data Recipes for Reasoning Models, 2025. URL https://arxiv.org/abs/2506.04178.

Hanxu Hu, Yuxuan Wang, Maggie Huan, Jannis Vamvas, Yinya Huang, Zhijiang Guo, and Rico Sennrich. DeReason: A Difficulty-Aware Curriculum Improves Decoupled SFT-then-RL Training for General Reasoning, 2026. URL https://arxiv.org/abs/2603.11193.

Jiahao Huang, Fei Cheng, Junfeng Jiang, and Akiko Aizawa. Tailoring the Curriculum: Student-Centered Reasoning Distillation via Dynamic Data-Model Compatibility, 2026. URL https://arxiv.org/abs/2605.29229.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code, 2024. URL https://arxiv.org/abs/2403.07974.

Feiyang Kang, Hanze Li, Adam Nguyen, Mahavir Dabas, Jiaqi W. Ma, Frederic Sala, Dawn Song, and Ruoxi Jia. Can Generalist Agents Automate Data Curation?, 2026. URL https://arxiv.org/abs/2606.04261.

Kimi Team. Kimi K3: Open Frontier Intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

Qizheng Li, Yifei Zhang, Xiao Yang, Xu Yang, Zhuo Wang, Weiqing Liu, and Jiang Bian. FT-Dojo: Towards Autonomous LLM Fine-Tuning with Language Agents, 2026. URL https://arxiv.org/abs/2603.01712.

Yang Li, Youssef Emad, Karthik Padthe, Jack Lanchantin, Weizhe Yuan, Thao Nguyen, Jason Weston, Shang-Wen Li, Dong Wang, Ilia Kulikov, and Xian Li. NaturalThoughts: Selecting and Distilling Reasoning Traces for General Reasoning Tasks, 2025. URL https://arxiv.org/abs/2507.01921.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's Verify Step by Step, 2023. URL https://arxiv.org/abs/2305.20050.

Joy Jia Yin Lim, Xin Huang, Hao Peng, Yaxi Lu, Xin Cong, Zhong Zhang, Maosong Sun, and Yankai Lin. What is Missing from AI Post-Training AI: An Empirical Analysis, 2026. URL https://arxiv.org/abs/2608.19072.

M-A-P Team. SuperGPQA: Scaling LLM Evaluation across 285 Graduate Disciplines, 2025. URL https://arxiv.org/abs/2502.14739.

Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhu Chen. General-Reasoner: Advancing LLM Reasoning Across All Domains, 2025. URL https://arxiv.org/abs/2505.14652.

Zerun Ma, Guoqiang Wang, Xinchen Xie, Yicheng Chen, He Du, Bowen Li, Yanan Sun, Wenran Liu, Kai Chen, and Yining Li. TREX: Automating LLM Fine-tuning via Agent-Driven Tree-based Exploration, 2026. URL https://arxiv.org/abs/2604.14116.

Huanzhi Mao, Raymond Tsao, Jingzhuo Zhou, Shishir G. Patil, and Joseph E. Gonzalez. BFCL V4: Web Search. Berkeley Function Calling Leaderboard official blog, 2025. URL https://gorilla.cs.berkeley.edu/blogs/15_bfcl_v4_web_search.html.

Mathematical Association of America. American Invitational Mathematics Examination 2025, 2025. URL https://maa.org/maa-invitational-competitions/.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, et al. Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Ö. Arık, and Tomas Pfister. MLE-STAR: Machine Learning Engineering Agent via Search and Targeted Refinement, 2025. URL https://arxiv.org/abs/2506.15692.

OpenAI. Introducing Codex. OpenAI, 2025. URL https://openai.com/index/introducing-codex/.

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. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing Verifiable Instruction Following. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2507.02833.

Qwen Team. Qwen3.5-4B. Hugging Face model card, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-4B.

Qwen Team. Qwen3.5-4B-Base. Hugging Face model card, 2026b. URL https://huggingface.co/Qwen/Qwen3.5-4B-Base.

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.

David Rein et al. GPQA: A Graduate-Level Google-Proof Q&A Benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. AI-Researcher: Autonomous Scientific Innovation, 2025. URL https://arxiv.org/abs/2505.18705.

Dingzirui Wang, Xuanliang Zhang, Keyan Xu, Qingfu Zhu, and Wanxiang Che. CurateEvo: Data-Curation Evolving for Agentic Post-Training, 2026a. URL https://arxiv.org/abs/2607.06140.

Haoqing Wang, Xiang Long, Ziheng Li, Yilong Xu, Tingguang Li, and Yehui Tang. To Mix or To Merge: Toward Multi-Domain Reinforcement Learning for Large Language Models, 2026b. URL https://arxiv.org/abs/2602.12566.

Yubo Wang et al. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark, 2024. URL https://arxiv.org/abs/2406.01574.

Changyi Xiao, Caijun Xu, and Yixin Cao. Reinforcement Learning with Conditional Expectation Reward, 2026. URL https://arxiv.org/abs/2603.10624.

Mingyu Xu, Cheng Fang, Keyue Jiang, Yuqian Zheng, Yanghua Xiao, Baojian Zhou, Qifang Zhao, Suhang Zheng, Xiuwen Zhu, Jiyang Tang, Yongchi Zhao, Yijia Luo, Zhiqi Bai, Yuchi Xu, Wenbo Su, Wei Wang, Bing Zhao, Lin Qu, and Xiaoxiao Xu. Logics-STEM: Empowering LLM Reasoning via Failure-Driven Post-Training and Document Knowledge Enhancement, 2026. URL https://arxiv.org/abs/2601.01562.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist-v2: Workshop-Level Automated Scientific Discovery via Agentic Tree Search, 2025. URL https://arxiv.org/abs/2504.08066.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling Data for Software Engineering Agents, 2025. URL https://arxiv.org/abs/2504.21798.

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

Taro Yano, Yoichi Ishibashi, and Masafumi Oyamada. LaMDAgent: An Autonomous Framework for Post-Training Pipeline Optimization via LLM Agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 30078–30095. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1529. URL https://aclanthology.org/2025.emnlp-main.1529/.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. LIMO: Less is More for Reasoning, 2025. URL https://arxiv.org/abs/2502.03387.

Jiachen Yu, Zhihao Xu, Junjie Wang, and Yujiu Yang. Think-with-Rubrics: From External Evaluator to Internal Reasoning Guidance, 2026a. URL https://arxiv.org/abs/2605.07461.

Zhaojian Yu, Penghao Yin, Shuzheng Gao, Shilin He, Kai Cai, and Xiao-Ping Zhang. AutoTrainess: Teaching Language Models to Improve Language Models Autonomously, 2026b. URL https://arxiv.org/abs/2606.31551.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, Zhaoyang Liu, Bolin Ding, and Jingren Zhou. AgentEvolver: Towards Efficient Self-Evolving Agent System, 2025. URL https://arxiv.org/abs/2511.10395.

Xueliang Zhao, Wei Wu, Jian Guan, Zhuocheng Gong, and Lingpeng Kong. PromptCoT 2.0: Scaling Prompt Synthesis for Large Language Model Reasoning, 2025. URL https://arxiv.org/abs/2509.19894.

Zhengyang Zhao, Shengjie Ye, Lu Ma, Hao Liang, Hengyi Feng, and Wentao Zhang. ANDES: Agent Native Data Evolving Synthesis Tool for Autonomous Instruction Alignment, 2026. URL https://arxiv.org/abs/2606.01279.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-Following Evaluation for Large Language Models, 2023. URL https://arxiv.org/abs/2311.07911.

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

A Appendix

A.1 Evaluation Splits and Metrics

Table 3 gives the fixed validation and test sizes. We allocate approximately 85% of each task to testing, rounding the test size down; TAU2 is split and rounded by domain, and BFCL by category. Optimization uses validation feedback only. Scores in the tables and trajectory figures use the held-out test split. The reported scores therefore describe our fixed splits, not the complete public benchmark test sets. Scores, means, and gains are computed before truncation to two decimal places.

Table 3: Validation and test sizes used in all comparisons. TAU2 is split by domain and BFCL by category.

Task	Validation	Test	Primary metric
MMLU-Pro	1,805	10,227	Accuracy
SuperGPQA	3,980	22,549	Accuracy
GPQA-Diamond	30	168	Accuracy
MATH-500	75	425	Accuracy
AIME 2025	5	25	Accuracy
IFEval	82	459	Strict prompt-level accuracy
IFBench	45	255	Prompt-level accuracy
LiveCodeBench v6	159	896	Pass@1, one response per problem
Terminal-Bench 2.0	14	75	Task success rate
TAU2-Bench	$8 + 18$	$42 + 96$	Mean of Airline and Telecom
BFCL-V4	690	3,890	Renormalized no-Web score

AIME uses 25 test questions and one greedy response per question. One additional correct answer changes a single-run score by four percentage points, or the two-run mean by two points. LiveCodeBench uses one response per problem. TAU2 equally averages Airline and Telecom scores over 42 and 96 tasks, respectively.

BFCL counts refer to unique scored examples; repeated Memory backends do not create additional test questions. Live Relevance diagnostics are excluded from these counts and from Overall (Appendix A.4).

A.2 Extended IFEval Experiment

This appendix supplements Section 4.3 with implementation details for the extended Qwen3.5-4B-Base run. C1–C8 follow the chronological order in Figure 3. They index candidate checkpoints, not eight accepted updates. This experiment is separate from the Qwen3.5-4B-Instruct ablation in Appendix A.3.

Data Sources and Preparation. The initial mixture draws on Tulu 3 Persona-IF, RubricHub's released best-of-six responses, and Nemotron Instruction-Following Chat v1. We use the selected RubricHub responses, not its intermediate candidate collection, and a filtered subset of Nemotron rather than its full post-training recipe. Later trials construct targeted examples and retain answers that pass the official executable instruction checks. These checks validate the specified constraints; they do not independently establish factual correctness or the quality of a reasoning trace.

Table 4 reports the final exported training-set sizes. Component counts refer to collections before final selection, deduplication, and overlap filtering, and therefore need not sum to the exported totals. C5 starts from C3 rather than the unsuccessful C4 checkpoint. It revises C4's data strategy while keeping the earlier model.

Table 4: Data preparation and training ancestry for the extended IFEval Base experiment. Counts in the last column are final exported examples.

Ckpt.	Parent	Strategy and data construction	Examples
C1	Initial Base	**Persona-conditioned constraint tuning.** Use released instruction–response pairs covering keyword, formatting, length, and capitalization constraints. Preserve message structure, validate formatting, deduplicate, and audit exact overlap with evaluation queries.	29,900
C2	C1	**Rubric-guided response tuning.** Combine persona-conditioned examples with approximately 26K released responses selected by best-of-six rubric scoring. Convert the selected pairs to the training message format before merging and filtering.	55,998
C3	C2	**Multi-source instruction mixture tuning.** Combine approximately 30K persona-conditioned, 24K rubric-selected, and 16K Nemotron examples. Retain non-thinking responses from the additional source, remove visual markers, and filter by length.	69,900
C4	C3	**Execution-verified targeted constraint tuning.** Mix approximately 23.4K verified synthetic examples with 13K replay examples. Target character and keyword frequencies, punctuation, capitalization, and sentence counts identified by the preceding error analysis.	36,299
C5	C3	**Replay-preserving additive constraint tuning.** Retain the full 69,900-example C3 mixture and add approximately 23,750 execution-verified examples, then deduplicate and filter. Unlike C4, this trial adds targeted supervision without replacing the broader mixture.	93,000
C6	C5	**Residual-error-guided constraint augmentation.** Extend the 93K-example mixture with 14,524 verified examples covering remaining lexical, counting, formatting, length, and punctuation errors.	107,000
C7	C6	**Explicit constraint reasoning with self-verification.** Combine 2,575 Qwen3.5-122B-generated reasoning traces with 18K verified answers augmented by rationales based on computable constraint checks. Retain examples whose final answers pass the checker.	20,000
C8	C7	**Precision-focused self-verification refinement.** Extend C7's 20,575-example pre-filter collection with 562 verified teacher-generated traces and 8K additional self-checking examples for exact-counting and lexical constraints, then filter the combined collection.	24,500

Training Configuration. All eight trials use full-parameter post-training for three epochs with a global batch size of 128. Table 5 lists the learning rates and sequence limits. The training sequence limit is distinct from the evaluation context window and maximum output length. The augmented self-checking examples should not be interpreted as independently sampled teacher reasoning traces.

Table 5: Training settings for the extended IFEval run. The sequence limit is measured in tokens.

Checkpoints	Learning rate	Sequence limit
C1–C3	1×10^{-5}	4,096
C4	5×10^{-6}	4,096
C5–C6	1×10^{-5}	4,096
C7–C8	1×10^{-5}	8,192

A.2.1 Evaluation and Resource Accounting

Generation and Scoring. The evaluation prompts and official strict prompt-level scoring procedure remain fixed. All checkpoints follow Section 4.1: a 128K context window, non-thinking generation, and greedy decoding with temperature 0. C7–C8 use reasoning-augmented training data but retain these evaluation settings.

Resource Budget. The extended experiment increases the time limit to 100 hours under the same resource allocation and timing rules as the main experiments, with the three-round limit removed. Checkpoints are equally spaced; their labels give elapsed time. Chronological order differs from training ancestry: C4 is an executed attempt even though C5 resumes from C3.

A.3 Ablation Implementation Details

This appendix supplements Section 4.4 with the replacements used for each mechanism and the recorded optimization decisions on five tasks. All runs start from Qwen3.5-4B-Instruct and follow the resource limits and end-to-end timing protocol in Section 4.1, stopping after 24 hours or three optimization rounds, whichever occurs first. The data sizes and training configurations in Appendix A.2 belong to the separate Base experiment and are not reused as descriptions of these ablations.

Module Replacements. The underlying agent retains access to tools, artifact inspection, and ordinary debugging in every variant. The replacements remove our prescribed procedures, not these general capabilities. Table 6 makes the distinction explicit.

Table 6: Basic alternatives used in the ablations. ER denotes Execution-Guided Refinement within Executable Model Updates.

Variant	Procedure omitted	Basic alternative retained
w/o EA	Joint metric- and case-level attribution with explicit focal-problem selection.	An unstructured summary of the evaluation results guides the next action.
w/o UD	Required candidate comparison, reproduction-difficulty assessment, resource verification, and outcome-aware reuse.	Ordinary research followed by adoption of a feasible method. The agent may still inspect resources or reconsider its choice.
w/o ER	Required sample-level checks, short training runs, and the execution-guided refinement procedure before full execution.	Ordinary execution and debugging within the same model-update stage. Repairs remain possible after problems are noticed.

Recorded Optimization Decisions. Table 7 summarizes the strategies and execution issues in each run. Steps denote successive recorded decisions, which can include data preparation, training, adapter rescaling, or implementation repair; they do not necessarily denote three newly trained checkpoints. Data counts are included where recorded and are specific to the model initialization used in that run.

Table 7: Recorded strategy and execution paths for the five Instruct-model ablation variants. These records provide context for the comparisons in Table 2.

Variant	Step 1	Step 2	Step 3
MMLU-Pro			
Full	Train on 20K long mathematical reasoning examples.	Train on 20K concise final-answer examples; repair end-of-sequence handling and premature termination.	Introduce cross-domain supervision using complete, verified teacher responses.
w/o EA	Train on 10K auxiliary knowledge questions with short rationales.	Expand to 18K examples and add output-format anchors.	Introduce structured short-form reasoning supervision.
w/o UD	Apply critique-guided distillation.	Switch to long-form mathematical reasoning data.	Introduce truth-teller/liar reasoning exercises.
w/o ER	Directly train on long mathematical reasoning traces.	Switch to final-answer supervision while termination and template issues remain unresolved.	Train on 215K examples without checking teacher-response completeness.

Continued on next page

Table 7 (continued)

Variant	Step 1	Step 2	Step 3
MATH-500			
Full	Train on 20K mathematical examples with final-answer supervision.	Incorporate 20K examples with extended reasoning.	Select 4,947 high-quality reasoning trajectories for further training.
w/o EA	Apply LoRA training on a general mathematical problem set.	Set the adapter scaling factor to 0.48.	Set the adapter scaling factor to 0.30.
w/o UD	Adopt final-answer training data.	Switch to a large mathematical instruction collection.	Return to long-form mathematical reasoning data.
w/o ER	Train on final-answer examples with limited configuration checks.	Apply a stronger LoRA update.	Reduce update strength as a corrective adjustment.
LiveCodeBench v6			
Full	Train on concise code solutions validated through execution.	Distill low-temperature reasoning responses selected for correctness.	Add higher-temperature, more diverse responses and replay data.
w/o EA	Apply LoRA training on concise code solutions.	Expand the same type of short-solution data.	Adjust sequence length and adapter strength.
w/o UD	Begin with general code instruction tuning.	Switch to difficult code-reasoning examples.	Mix code-reasoning examples with general code replay.
w/o ER	Directly use code examples with long reasoning traces.	Proceed with unresolved thinking-template and termination checks.	Partially repair generation and execution settings in the training-data pipeline.
TAU2-Bench			
Full	Train on 12K verifiable tool-use trajectories.	Introduce airline trajectories emphasizing evidence-supported write operations.	Repair the training-data user simulator and add 680 successful telecom trajectories.
w/o EA	Train on general verifiable tool-use trajectories.	Continue with airline-domain data.	Add retail-domain trajectories.
w/o UD	Begin with basic tool-trajectory training.	Generate 88 telecom-domain trajectories.	Switch to airline write-operation trajectories.
w/o ER	Select telecom-domain trajectory generation.	Encounter a training-data simulator unable to invoke user-side tools.	Partially repair deduplication and schema issues.
IFEval			
Full	Train on persona-conditioned requests with verifiable constraints.	Add rubric-selected best-of-six responses.	Expand to a three-source mixture with conversational instruction data.
w/o EA	Begin with persona-conditioned instruction tuning.	Expand general instruction-following data.	Continue mixing general instruction data.
w/o UD	Begin with targeted constraint examples.	Switch to a general instruction-following mixture.	Adopt an executable data-synthesis strategy without structured comparison.
w/o ER	Generate targeted constraint examples.	Replace the broader mixture with specialized data.	Continue with limited replay coverage and small-scale validation.

Optimization Focus and Strategy Selection. The MATH-500 run without EA moves from general mathematical training to two adapter-scaling adjustments, whereas the full harness changes the supervision and then selects reasoning trajectories. The variant without UD moves between final-answer supervision, a large instruction collection, and long-form reasoning. These decisions illustrate different uses of the available budget. A strategy change is not itself a failure; the comparison concerns how each run diagnoses its needs and selects the next action.

Execution Checks and Recovery. The variants without ER still make corrective adjustments, including adapter rescaling, configuration changes, and schema repairs. Their records also contain unresolved termination behavior, unchecked response completeness, and simulator-tool failures. The LiveCodeBench configuration changes and TAU2 simulator repairs concern training-data generation and execution debugging; all models use the same fixed final evaluation environment. Our mechanism organizes checks before full execution, rather than introducing debugging as a new capability. The records help interpret the observed outcomes, but do not establish that any one implementation issue alone caused a particular score change.

A.4 BFCL Scoring Without Web Search

The official BFCL-V4 score assigns 40% to Agentic, which equally averages Web Search and Memory (Mao et al., 2025). Counting omitted Web Search as zero limits Overall to 80%. For our no-Web metric, we remove its 20% weight and renormalize the remaining components:

$$J_{\text{no-web}} = \frac{0.1J_{\text{non-live}} + 0.1J_{\text{live}} + 0.1J_{\text{irr}} + 0.3J_{\text{multi}} + 0.2J_{\text{memory}}}{0.8}.$$

This gives weights of 12.5% each to Non-Live, Live, and Irrelevance, 37.5% to Multi-Turn, and 25% to Memory. We retain the original within-section aggregation, including equal averaging across the three Memory backends. This subset metric is not directly comparable to the official full-task Overall score.

We allocate 85% of each lowest-level category to held-out testing, with sample counts rounded down. The split contains 3,890 scored unique examples and 4,152 scored executions, since the same 131 Memory questions use three backends. Another 13 Live Relevance examples are diagnostic and excluded from Overall. Category-wise rounding can yield a different total from rounding the combined dataset. Shared Memory question IDs are assigned to a split before backend execution.

A.5 Limitations and Future Work

Following prior work on task-specific post-training (Rank et al., 2026; Li et al., 2026), we focus on target-task performance. We do not systematically measure whether specialization affects general capabilities, given the substantial evaluation effort and the many combinations of tasks, models, and training strategies. The substantial task-specific gains support applications that need focused model adaptation without repeated manual experimentation. Future work will study how to preserve general capabilities during this process.

We also restrict executable updates to supervised post-training. Extending the harness to reinforcement learning and broader post-training objectives is left to future work.