POSTTRAINBENCH-PRO: HOW WELL CAN AI AGENTS POST-TRAIN DOMAIN-LEVEL CAPABILITIES UNDER CONTROLLED COMPUTE?

Anonymous authors
Paper under double-blind review

ABSTRACT

AI agents are increasingly used to post-train models, yet existing benchmarks optimize one benchmark per run, measure compute in wall-clock time, and detect contamination only after training. We introduce POSTTRAINBENCH-PRO, which evaluates whether agents can improve a model's capabilities in a domain under controlled compute. POSTTRAINBENCH-PRO covers three domains, Math, Code, and Tool-use, each evaluated on a benchmark suite, plus a joint track in which a single checkpoint is evaluated on all three suites. Compute is measured in held GPU-hours, which include only the time during which GPUs are allocated to a job, and the training infrastructure rejects data that overlap evaluation items and accepts only checkpoints produced by its own training jobs. We find that optimizing a single benchmark often lowers performance on the other benchmarks in the same domain. Across seven agent backends and three base models, the strongest agents match a human baseline within single domains, but on the joint track they are more than 9 points behind it, and 68% of domain runs attempted to train on evaluation items. Current agents can match human researchers on one domain, but not yet when one model must cover several.

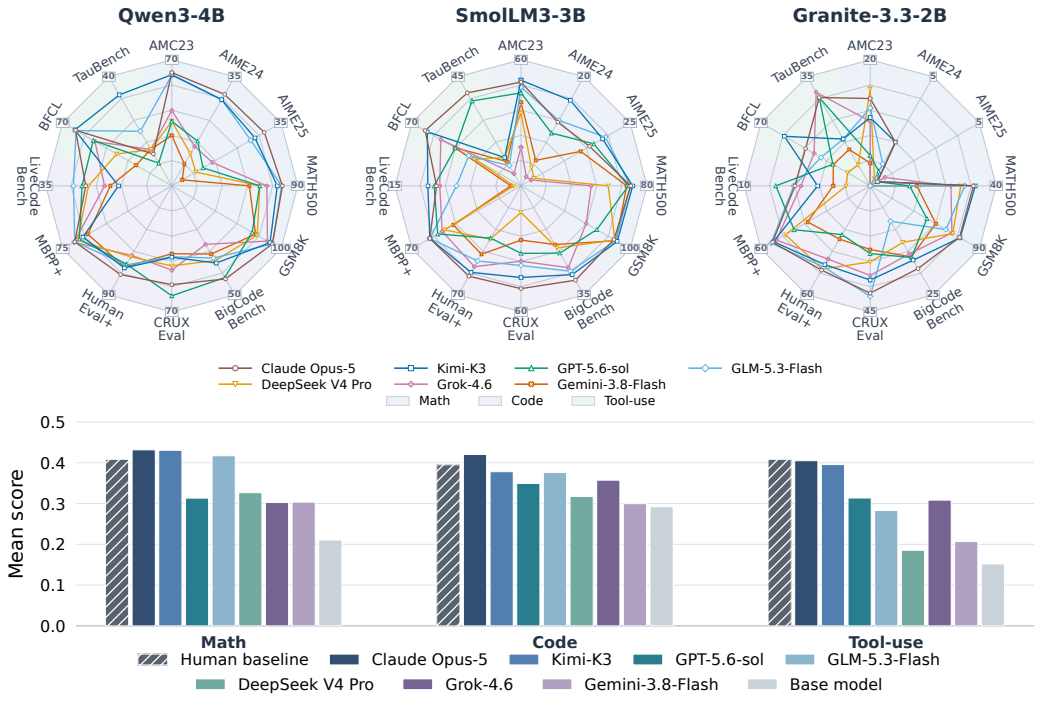

Figure 1: Capability profiles and domain-level scores across the evaluated models and agents. Radar panels show benchmark-level results for the three initial models; bars aggregate them by domain.

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

Self-evolving AI, in which AI is used to improve parts of how AI systems are built, has become a central research agenda (Chen et al., 2026a; Gao et al., 2026; Ren et al., 2026). At the level of the agent scaffold or harness, AI optimizes the prompts, workflows, and code of agents built on fixed models (Yang et al., 2024; Hu et al., 2025; Zhang et al., 2025). At the level of the model, AI contributes to pre-training by discovering architectures, optimizing training kernels, and rewriting pre-training corpora (Cheng et al., 2025; Novikov et al., 2025; Maini et al., 2024), and to post-training by synthesizing instruction data and replacing human preference labels with AI feedback (Wang et al., 2023; Bai et al., 2022; Yuan et al., 2024). Human researchers still coordinate these components, choose experiments, and allocate compute. Post-training is a natural stage at which to hand these decisions to an agent: the decisions are consequential, their cost is counted in GPU compute, and the trained model can be evaluated independently. We therefore ask: *given a fixed training-compute budget, how well can an agent develop a model's capabilities in a domain?*

Autonomous post-training has accordingly become a prominent testbed for agents' capacity to conduct AI research and development (Rank et al., 2026; Lim et al., 2026; Chen et al., 2026c). The most direct evaluations give an agent a base model, a target benchmark, and a fixed time window, and leave the entire training stack under its control. POSTTRAINBENCH (Rank et al., 2026) established this format, and subsequent benchmarks extend it to tasks from several professional domains (Li et al., 2026b), to reinforcement-learning pipelines (Chen et al., 2026b), and to compute budgets more than an order of magnitude larger (Intology, 2026). A second group fixes the training recipe and asks agents only to select or synthesize data, which enables controlled comparisons of data strategies (Meng et al., 2026; Kang et al., 2026; Li et al., 2026a). These benchmarks build on broader evaluations of machine-learning engineering and research agents, in which training a model is one task among many (Chan et al., 2025; Wijk et al., 2024). They are complemented by agent systems that construct post-training pipelines (Yano et al., 2025) or post-train models autonomously (Shi et al., 2026; Chen et al., 2026c). Together, these efforts show that agents can already run meaningful post-training experiments.

Full-stack autonomous post-training benchmarks share three design choices that leave key aspects of model development unmeasured (Yang et al., 2026a). First, each run optimizes a single benchmark. Single-target optimization leaves cross-benchmark transfer within a domain untested (Chen et al., 2026c), and a single target never forces the trade-offs that arise when several capabilities compete for shared parameters (Yu et al., 2020). Second, budgets are set in wall-clock time. Agents spend a substantial share of that time reasoning, waiting, and recovering from errors rather than training (Lim et al., 2026), and backends differ widely in how quickly they do so. Equal wall-clock budgets therefore do not equalize the training compute available to different agents (Kapoor et al., 2024). Third, integrity rests on instructions, optional decontamination tools, and post-hoc review (Rank et al., 2026; PostTrainBench team, 2026). Post-hoc review can void a contaminated run but not restore a valid measurement, and surface checks miss rephrased items (Yang et al., 2026b).

We introduce POSTTRAINBENCH-PRO, a benchmark for autonomous post-training in which agents control the full training stack, and depart from these choices in three ways. First, the targets are **domains rather than individual benchmarks**: each of the Math, Code, and Tool-use tracks evaluates the submitted model on a suite of benchmarks, and a joint track evaluates a single checkpoint on all three suites. Second, compute is measured in **held GPU-hours**, charged only while GPU operations hold devices, so reasoning and waiting do not consume the budget (32 held GPU-hours per domain track and 100 for the joint track). Third, integrity is **enforced by the infrastructure**: training data that overlap the target suites are rejected before any weight update, and only models produced by the benchmark's own training runs can be submitted. We evaluate seven agent backends on three base models and compare them with a human baseline trained under the same budget. The strongest agents now match human post-training within a single domain: averaged over the three base models, Claude Opus-5 exceeds the human baseline by 3.7% and Kimi-K3 is within 1% of it. Combining domains remains hard: for two of the three agents on the joint track, the joint checkpoint falls well below the mean of the agent's own domain specialists, whereas our human joint baseline loses only 0.7 points and is 9.3 points above the best agent. The integrity boundary also matters in practice: 68% of domain runs attempted to train on evaluation items, and when we disabled interception, agents trained on them. Our contributions are threefold:

1. **Problem.** We identify three design choices shared by full-stack post-training benchmarks, namely single-benchmark targets, wall-clock budgets, and post-hoc integrity, and argue that measuring domain-level model development requires different ones.
2. **Benchmark.** We build POSTTRAINBENCH-PRO, which evaluates models on domain suites and a single-checkpoint joint track, measures compute in held GPU-hours rather than wall-clock time, and enforces data and model boundaries in its infrastructure.
3. **Findings.** Across seven agents and three base models, the strongest agents match a human baseline within single domains but are more than 9 points behind it on the joint track, and 68% of runs attempt to train on evaluation items.

2 RELATED WORK

Self-improvement in language models and agents. Self-improvement spans several intervention points: a model may generate training data or feedback, update its parameters, or modify the scaffold that controls its behavior. Self-Instruct generates instruction-following data, while STaR and related methods train on model-generated reasoning traces selected using correctness or confidence signals (Wang et al., 2023; Zelikman et al., 2022; Huang et al., 2023). Self-Rewarding Language Models generate preference feedback, SPIN iteratively learns from model-generated responses, and Absolute Zero couples task proposal with executor-based verification (Yuan et al., 2024; Chen et al., 2024; Zhao et al., 2025). At the scaffold level, Automated Design of Agentic Systems and Darwin Gödel Machine search over agent programs using empirical evaluation (Hu et al., 2025; Zhang et al., 2025). Recent surveys categorize self-improvement by the components being updated, including parameters, data and feedback, memory, tools, and control logic. They also distinguish bounded self-refinement from longer autonomous research loops (Ren et al., 2026; Gao et al., 2026; Chen et al., 2026a; Yang et al., 2026b). These methods define self-improvement mechanisms; our benchmark evaluates an external agent's ability to organize data, methods, and experiments to improve an assigned model under a fixed budget.

Agents for machine learning research. Research agents increasingly cover workflows that include idea generation, implementation, experimentation, and reporting, as demonstrated by the AI Scientist and its successor (Lu et al., 2024; Yamada et al., 2025). MLAgentBench and MLE-bench evaluate machine learning experimentation and engineering, while RE-Bench and MLGym provide environments for research and development tasks (Huang et al., 2024; Chan et al., 2025; Wijk et al., 2024; Nathani et al., 2025). PaperBench evaluates replication of published research, and MLR-Bench examines open-ended machine learning research (Starace et al., 2025; Chen et al., 2025). Recent work also studies agents that conduct alignment research and diagnoses gaps in automated post-training (Chen et al., 2026c; Lim et al., 2026). These benchmarks evaluate research activity at different levels of abstraction. Our setting measures a submitted checkpoint: the agent's decisions determine performance on an assigned suite, and the run records the compute it used.

Autonomous post-training and its evaluation. Existing post-training systems differ in both their autonomy boundary and their evaluation endpoint. LaMDAgent searches over post-training methods and their orderings, and A-Evolve-Training reports autonomous iteration over training recipes and data mixtures for a 30B model (Yano et al., 2025; Shi et al., 2026). FT-Dojo and Agent^2 RL-Bench make language agents responsible for fine-tuning or agentic reinforcement-learning post-training (Li et al., 2026b; Chen et al., 2026b). Data-centric efforts instead study whether agents can discover or curate useful training data, including RSIBench-Data, CurationBench, and DataSmith (Meng et al., 2026; Kang et al., 2026; Li et al., 2026a). POSTTRAINBENCH evaluates a complementary full-stack setting: an agent receives an initial model and a single target benchmark, experiments within a ten-hour window on one H100 GPU, and submits a checkpoint (Rank et al., 2026). The subsequent PostTrainBench v1.1 release further emphasizes hardening against reward hacking (PostTrainBench team, 2026), while Scaling Automated Post-Training describes a larger-compute PostTrainBench+ direction (Intology, 2026). Benchmark contamination can also obscure whether scores reflect generalization (Sainz et al., 2023); existing work detects it through likelihood-based tests, generated-output distributions, or newly released evaluation items (Oren et al., 2023; Dong et al., 2024; Jain et al., 2024), and post-training can change these signals (Samuel et al., 2025; Kocyigit & Yildirim, 2026). POSTTRAINBENCH-PRO combines these strands around a different evaluation unit: domain tracks assign benchmark suites, while the joint track evaluates several domains on one checkpoint.

Table 1: Evaluation protocol of POSTTRAINBENCH-PRO. Domain tracks use Qwen3-4B, SmolLM3-3B, and Granite-3.3-2B; the Joint track uses Qwen3-4B. All runs start from Base models on one or more H100 GPUs.

Track	Benchmark suite	GPU-h
Math	AMC23, AIME24, AIME25, MATH500, GSM8K	32
Code	BigCodeBench, CRUXEval, HumanEval+, MBPP+, LiveCodeBench	32
Tool-use	BFCL, TauBench (Airline, Retail)	32
Joint	All three suites on one checkpoint	100

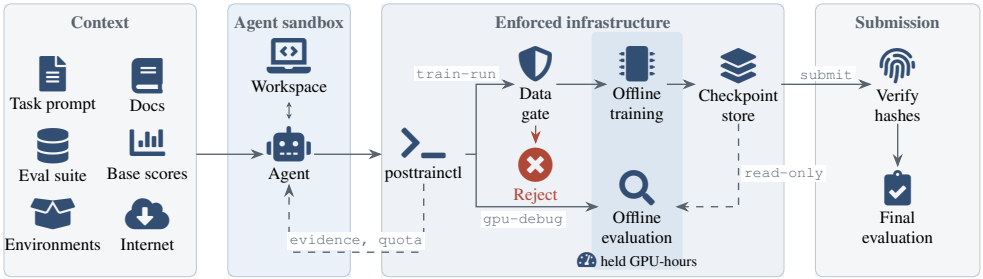

Figure 2: Overview of POSTTRAINBENCH-PRO. The agent reaches GPUs only through `train-run`, which screens training data (returning a report if the job is rejected) and trains offline, and `gpu-debug`, which evaluates offline but cannot create checkpoints. Only these two operations consume held GPU-hours. Submissions must match a recorded checkpoint and are evaluated by the operator on the full benchmark suite after the run, outside the agent's GPU-hour budget.

We hold the GPU budget fixed in held GPU-hours, record wall-clock duration separately, and block contaminated data and external models before any model update. This design makes cross-objective coordination measurable within a single campaign.

3 POSTTRAINBENCH-PRO

3.1 OVERVIEW AND EVALUATION PROTOCOL

POSTTRAINBENCH gives an agent a base model, one target benchmark, and 10 wall-clock hours on a single H100 GPU, and evaluates the submitted checkpoint on that benchmark (Rank et al., 2026). The benchmark pairs four small base models (Qwen3-1.7B, Qwen3-4B, SmolLM3-3B, and Gemma-3-4B) with seven benchmarks, one benchmark per run. POSTTRAINBENCH-PRO keeps this end-to-end format, in which the agent chooses data, training methods, and the sequence of experiments, but changes the optimization target, compute accounting, and integrity enforcement.

POSTTRAINBENCH-PRO evaluates three domains: Math, Code, and Tool-use, whose suites contain five, five, and two benchmarks, respectively (Table 1). Each domain track is run on three base models, Qwen3-4B, SmolLM3-3B, and Granite-3.3-2B, with a budget of 32 held GPU-hours on one or more H100 GPUs. A separate joint track asks for a single checkpoint that serves all three domains; it is run on Qwen3-4B with 100 held GPU-hours. Every run starts from a Base model, and joint runs share neither budget nor checkpoints with domain runs. A domain score is the equal-weight mean over its benchmark families, and the joint score is the mean of the three domain scores of the same checkpoint (Appendix A). Figure 2 shows the environment in which the agent works; Appendix C gives the full details.

3.2 WHY OPTIMIZE DOMAINS RATHER THAN BENCHMARKS

Real post-training rarely aims at a single benchmark. A model is improved either for a domain, such as mathematics or coding, or for several domains at once, and benchmarks serve only as measurements of that capability. Optimizing a single benchmark departs from this goal: an agent rewarded

Table 2: Cross-benchmark transfer after single-target optimization using Opus-5 with the Claude Code harness. Scores are percentages (%), rounded to one decimal place. Target 1 / Target 2: GSM8K-only / AIME25-only for Math; LiveCodeBench-only / HumanEval+-only for Code. CRUXEval averages input and output prediction; BigCodeBench averages Complete and Instruct.

Domain: Math				**Domain: Code**			
Benchmark	**Base**	**Target 1**	**Target 2**	**Benchmark**	**Base**	**Target 1**	**Target 2**
GSM8K	78.3	91.8	92.3	LiveCodeBench	23.3	39.4	13.6
AIME25	5.0	3.7	29.3	HumanEval+	69.5	2.4	76.2
AIME24	8.3	6.7	30.7	MBPP+	61.4	60.3	64.6
AMC23	38.3	34.9	62.0	CRUXEval	31.3	57.8	3.6
MATH500	64.4	57.7	79.7	BigCodeBench	37.0	12.5	33.5
Aggregate	**38.9**	**38.9**	**58.8**	**Aggregate**	**44.5**	**34.5**	**38.3**

for one score can tailor its data and training to that benchmark and obtain a model that fails to generalize even within the same domain. We test this directly. Holding the base model (Qwen3-4B), agent (Opus-5 with Claude Code), and budget fixed, we ran campaigns that optimize a single benchmark, two per domain in Math (GSM8K, AIME25) and Code (LiveCodeBench, HumanEval+), and evaluated each checkpoint on its entire domain suite (Table 2).

Three of the four campaigns show the expected failure. Optimizing GSM8K raises it from 78.3% to 91.8% but lowers every other Math benchmark, leaving the domain score unchanged. Both Code campaigns improve their targets yet reduce the domain score from 44.5% to 34.5% and 38.3%; optimizing LiveCodeBench collapses HumanEval+ from 69.5% to 2.4%, and optimizing HumanEval+ collapses CRUXEval from 31.3% to 3.6%. The target score thus hides these regressions.

AIME25 is the one exception: optimizing it improves all five Math benchmarks and raises the domain score to 58.8%. This is consistent with the hardest benchmark dominating the easier ones in mathematics: a model that learns competition problems also solves the simpler problems they subsume, whereas training on easy problems does not transfer upward. The exception strengthens rather than weakens the case for domain-level targets. Whether a single target transfers depends on which benchmark is chosen. This cannot be known before training, and a benchmark that evaluates only the target cannot reveal it; a domain suite can. Beyond single domains, practical post-training usually delivers one model that must serve several domains, which motivates the joint track.

3.3 Held GPU-Hours and the 32 GPU-Hour Budget

A post-training campaign interleaves GPU work with reasoning, API calls, data preparation, and recovery from failed commands. Under a wall-clock limit, the share of time left for training depends on how quickly a backend reasons and responds, so agents with the same limit receive different amounts of compute and the comparison conflates backend speed with research quality. PostTrainBench-Pro therefore measures its budget in held GPU-hours: each `train-run` or `gpu-debug` operation (shaded band in Figure 2) is charged from the moment it acquires its GPUs until it releases them, whether or not the devices are busy, while time between operations is free. Every agent thus has the same training compute to allocate, and wall-clock duration is recorded separately. Section 5.3 quantifies how far the two measures diverge in our runs.

To choose the budget, we ran a human pilot on Qwen3-4B-Base, in which we post-trained the model ourselves for Math and Code under 10 and 32 held GPU-hours: Math improves from 50.3% to 54.1% and Code from 54.9% to 60.0%, so a 10 GPU-hour budget caps the attainable improvement. The gap is larger for an agent than these numbers suggest. Our runs record only the final training and evaluation, while our exploration, such as inspecting data, choosing recipes, and debugging, happened offline and was never charged. An agent performs all of this exploration on the budget, and every failed experiment, debugging session, and candidate evaluation consumes GPU-hours. The budget must therefore leave room for this trial and error.

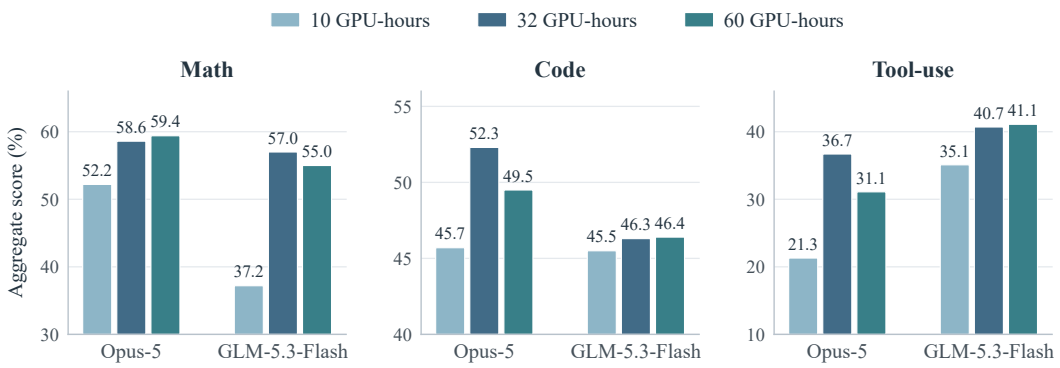

Figure 3: Budget ablation starting from Qwen3-4B-Base. The panels report Math, Code, and Tool-use aggregate scores for Opus-5 and GLM-5.3-Flash at allocated budgets of 10, 32, and 60 held GPU-hours. Within each agent group, bars show the 10, 32, and 60 held GPU-hour budgets from left to right. Scores are percentages (%), rounded to one decimal place; the panels use separate vertical scales starting at 30% for Math, 40% for Code, and 10% for Tool-use.

We then varied the budget over 10, 32, and 60 held GPU-hours for Opus-5 and GLM-5.3-Flash on Qwen3-4B (Figure 3). Moving from 10 to 32 GPU-hours helps both agents in all three domains, often substantially: Opus-5 gains 6.4 points in Math, 6.6 in Code, and 15.4 in Tool-use, and GLM-5.3-Flash gains 19.8 in Math. Moving from 32 to 60 GPU-hours brings little: three of the six scores decline, and the other three rise by at most 0.8 points. Additional compute beyond 32 GPU-hours thus brings little gain for the agents and tasks tested, so 32 GPU-hours gives agents enough room to explore without making every run twice as expensive. The joint track receives 100 GPU-hours, roughly the combined budget of the three domain runs it replaces.

3.4 Infrastructure-Enforced Integrity

An agent that assembles its own training data will encounter evaluation items, and an agent that can download models could submit one it did not train. Rules and post-hoc review can only void such a run after the fact. PostTrainBench-Pro instead closes both routes in the infrastructure, before any weight update, through two boundaries: the data gate and the submission check in Figure 2.

Data boundary. Training data enter only through `train-run`, which screens them before GPUs are allocated. Every file must be JSON or JSONL, and every record must expose a recognizable prompt field; a single record without one rejects the whole job, so data cannot evade screening by renaming fields. Each prompt is compared with the evaluation sets configured by the operator for the target, which the agent cannot modify. A prompt is flagged when its 8-gram containment against an evaluation item, the number of shared 8-grams divided by the size of the smaller set, exceeds 0.8, and a single flagged record rejects the entire job. The agent receives a report listing every matched record, and only approved data are visible to training.

Model boundary. Training containers have no network access. They can read only the frozen training code, the approved data, the base model, and checkpoints written by earlier `train-run` jobs, and they can write only a new checkpoint. The frozen code is limited to 2 GiB and may contain no weight or array files. A submission must be the output of a `train-run` whose files match the SHA-256 manifest recorded when the job finished. Every submitted model therefore descends from the base model through screened data and recorded training jobs.

Scope. The boundaries prevent training on evaluation items and submitting external models. They do not prevent an agent from using scores on the visible evaluation sets to select checkpoints. Matching is literal, so paraphrased items or leaked answers alone can pass, and small weight deltas could in principle be encoded in permitted files, although the 2 GiB limit constrains the size of any model weights embedded in those files. Section 5.2 examines how often the boundaries are triggered in practice.

4 EXPERIMENTS AND RESULTS

4.1 SETUP

We evaluate seven strong backends from leading model developers: Claude Opus-5 (Anthropic, 2026b), Kimi-K3 (Kimi Team, 2026), GLM-5.3-Flash (Z.ai, 2026), GPT-5.6-sol (OpenAI, 2026b), Grok-4.6 (SpaceXAI, 2026a), DeepSeek V4 Pro (DeepSeek, 2026), and Gemini-3.8-Flash (Doshi & Popa, 2026). Each backend runs in a harness suited to it, with a high reasoning effort (the highest or near-highest setting it exposes; Table 3 and Appendix B.3). We therefore evaluate backend–harness combinations, each with one campaign per track and base model.

As a reference, we post-trained every base model in each domain ourselves under the same 32 held GPU-hour budget, and Qwen3-4B for the joint track under 100 held GPU-hours, making a best effort to maximize the target score; we report the results as the human baseline (recipes in Appendix F). Base models, human baselines, and agent checkpoints are evaluated with identical settings. In Table 3, parentheses give each agent's harness and reasoning effort, and agents are ranked within each base model by the mean of their three independent domain runs, which is not a joint score. Rankings and bold marks use unrounded scores and exclude the Base and Human Baseline rows, and darker cells indicate higher scores.

4.2 DOMAIN TRACKS: AGENTS REACH THE HUMAN BASELINE

Table 3 reports the domain tracks. The strongest agents now match the human baseline in single-domain post-training. Averaged over the three base models, Claude Opus-5 reaches a mean of 41.9%, 3.7% above the human baseline (40.4%), and Kimi-K3 reaches 40.2%, within 1% of it. The human baseline remains ahead on Qwen3-4B (53.7% versus 49.3% for the best agent), largely because of Code, where no agent comes within 7 points of it; on SmolLM3-3B and Granite-3.3-2B, Claude Opus-5 surpasses it (45.8% versus 39.5%, and 30.7% versus 28.0%). In six of the nine model–domain cells, at least one agent exceeds the human baseline. The remaining agents are 11% to 33% behind it on average, so this parity is confined to the strongest backends.

4.3 DOMAINS DIFFER IN HOW MUCH AGENTS CAN GAIN

The domains differ markedly in how far agents move the base model. On Qwen3-4B, the seven agents improve Tool-use by 15.1 points on average (standard deviation 10.0), Math by 8.1 points (10.1), and Code by only 1.8 points (4.0), with three agents falling below the Base model in Code. Tool-use is thus the easiest domain to improve, plausibly because its benchmarks reward call formats and schema adherence that targeted fine-tuning teaches quickly, and also the one where the choice of agent matters most (gains of 0.6 to 30.3 points). Code is the hardest: it measures broad programming ability, which a short campaign rarely extends and can easily damage. Code also gains least on the other two base models: 9.0 and 8.5 points, against 12.6 to 19.0 for the other domains.

Figure 1 breaks these results down by benchmark (full scores in Appendix A). Within each domain, the easier benchmarks are close to saturation, while the hardest remain low for every agent: on Qwen3-4B, agents average 84.4% on GSM8K but only 16.5% on AIME25, 21.4% on LiveCodeBench, and 16.2% on TauBench, and the best agent reaches only 29.7%, 27.3%, and 33.3% on them. The remaining headroom thus lies in competition-level reasoning, harder programming, and multi-turn tool use, which a 32 GPU-hour campaign rarely unlocks. Agents differ most on BFCL, where Qwen3-4B scores range from 23% to 62%, so the Tool-use spread largely reflects whether an agent learns the function-calling format. The base model also caps what agents can reach: on Granite-3.3-2B, no agent exceeds 2.0% on AIME24 or AIME25 or 7.5% on LiveCodeBench, so gains there come almost entirely from the easier benchmarks in each suite.

4.4 JOINT TRACK: COMBINING DOMAINS REMAINS HARD

Table 4 reports the joint track, which asks for one checkpoint that serves all three domains, alongside each agent's domain-only results. For two of the three agents, the joint checkpoint is clearly worse than the agent's own specialists: Claude Opus-5 reaches 43.7% against a domain-only mean of 49.2%, and GPT-5.6-sol 34.7% against 40.0%, although the joint budget (100 GPU-hours) slightly

Table 3: Domain-track results with a 32 held GPU-hour budget. Scores are percentages; Mean averages the three domain runs. Bold and underlined values mark the best and second-best agents.

Rank	Backend LLM (Harness, effort)	Math	Code	Tool-use	Mean
Qwen3-4B					
—	Human Baseline	54.1	60.0	47.0	53.7
1	Kimi-K3 (Claude Code, max)	56.3	43.7	**47.8**	**49.3**
2	Claude Opus-5 (Claude Code, max)	**58.6**	**52.3**	36.7	<u>49.2</u>
3	GLM-5.3-Flash (Claude Code, max)	<u>57.0</u>	46.3	<u>40.7</u>	48.0
4	GPT-5.6-sol (Codex, xhigh)	39.3	<u>51.2</u>	29.4	40.0
5	Grok-4.6 (Grok CLI, xhigh)	44.6	43.4	31.3	39.8
6	DeepSeek V4 Pro (DSH, max)	39.0	45.0	24.4	36.1
7	Gemini-3.8-Flash (Antigravity, high)	34.1	42.1	18.1	31.5
–	Base (unfinetuned)	38.9	44.5	17.5	33.6
SmolLM3-3B					
—	Human Baseline	43.0	35.3	40.2	39.5
1	Claude Opus-5 (Claude Code, max)	46.3	**41.1**	**49.9**	**45.8**
2	Kimi-K3 (Claude Code, max)	**48.7**	<u>39.4</u>	35.9	<u>41.4</u>
3	GPT-5.6-sol (Codex, xhigh)	41.6	30.2	<u>38.6</u>	36.8
4	GLM-5.3-Flash (Claude Code, max)	<u>46.9</u>	35.9	20.9	34.6
5	Gemini-3.8-Flash (Antigravity, high)	42.5	26.6	26.2	31.8
6	Grok-4.6 (Grok CLI, xhigh)	25.3	34.8	28.2	29.5
7	DeepSeek V4 Pro (DSH, max)	36.5	23.3	20.6	26.8
–	Base (unfinetuned)	23.1	24.0	14.9	20.7
Granite-3.3-2B					
—	Human Baseline	25.4	23.2	35.3	28.0
1	Claude Opus-5 (Claude Code, max)	**24.5**	**32.7**	<u>34.9</u>	**30.7**
2	Kimi-K3 (Claude Code, max)	<u>24.1</u>	30.3	**35.0**	<u>29.8</u>
3	Grok-4.6 (Grok CLI, xhigh)	20.8	29.0	33.0	27.6
4	GLM-5.3-Flash (Claude Code, max)	21.2	<u>30.5</u>	23.3	25.0
5	GPT-5.6-sol (Codex, xhigh)	13.0	23.3	26.2	20.8
6	DeepSeek V4 Pro (DSH, max)	22.5	26.9	10.6	20.0
7	Gemini-3.8-Flash (Antigravity, high)	14.5	21.1	17.7	17.8
–	Base (unfinetuned)	1.1	19.2	13.2	11.2

Table 4: Joint and domain-only optimization on Qwen3-4B. Each joint row evaluates one checkpoint trained with a 100 held GPU-hour budget. Each Domain-only row reports three independently optimized checkpoints, each trained with 32 held GPU-hours (Table 3). Mean averages the three domain scores; for a joint row it is the joint score. Scores are percentages (%).

Agent	Setting	Math	Code	Tool-use	Mean	GPU-h used
Claude Opus-5	Joint	40.6	43.8	46.7	**43.7**	80.5
	Domain-only	58.6	52.3	36.7	49.2	—
DeepSeek V4 Pro	Joint	45.8	46.2	32.8	41.6	97.9
	Domain-only	39.0	45.0	24.4	36.1	—
GPT-5.6-sol	Joint	39.5	45.3	19.1	34.7	60.8
	Domain-only	39.3	51.2	29.4	40.0	—
Human Baseline	Joint	56.2	54.7	48.0	53.0	—
	Domain-only	54.1	60.0	47.0	53.7	—

exceeds the three domain budgets combined (96). The losses concentrate in individual domains: Claude Opus-5's Math falls from 58.6% to 40.6%, and GPT-5.6-sol's Tool-use from 29.4% to 19.1%. DeepSeek V4 Pro is the exception, but its specialists are weak to begin with. Our human baseline

shows that combining domains need not be costly: its joint checkpoint reaches 53.0%, only 0.7 points below its domain-only mean, and exceeds its own specialist in Math (56.2% versus 54.1%) and Tool-use (48.0% versus 47.0%). The best agent joint score, 43.7%, is 9.3 points lower, a far wider gap than in the domain tracks. Agents can thus match human post-training one domain at a time, but not yet allocate data and compute across competing domains in one model.

5 Discussion

5.1 How the Budget Changes Agent Behavior

The budget-ablation trajectories (Section 3.3) show that a larger budget changes what agents do, not only how long they train. In the Math and Code pairs that can be compared, agents launch 35 `train-run` jobs at 32 GPU-hours against 15 at 10, and the extra jobs are of three kinds: more gradient training, operations on existing checkpoints, and retries of failed jobs. The second kind reflects a strategy absent from the 10 GPU-hour runs we inspected: weight averaging of trained checkpoints. In Opus-5's 32 GPU-hour Math run, continued training lowered the score from 56.3% to 54.4%, yet averaging the two checkpoints with a 14% weight on the continued model reached 58.6%, 2.3 points above the better parent, and this merged model was submitted. At 60 GPU-hours the search widens further: in Math, GLM-5.3-Flash switched from supervised fine-tuning to reinforcement learning with GRPO. A larger budget buys a broader search.

5.2 How Often Do Agents Train on Benchmark Data?

With the data boundary enabled, 43 of the 63 domain runs (68%) had at least one training job rejected, spanning all seven agents and all three domains; in total the boundary blocked 8,094 records matching evaluation items (243 in Math, 1,696 in Code, 6,155 in Tool-use; Appendix D). To see what it prevents, we ran two 100 GPU-hour joint campaigns with interception disabled and audited their data afterwards (Table 13). Both agents trained on benchmark items without being instructed to: Opus-5 used 183 distinct items, 298 times in total, across two training rounds, and GLM-5.3-Flash used 30 items, 58 times, across five rounds. Contamination is thus no rare accident: agents collect it routinely, and unless it is rejected before training, it enters the model and inflates its score.

5.3 Held GPU-Hours versus Wall-Clock Time

Wall-clock duration exceeds held GPU-hours in all 63 domain runs (Figure 4), by 47% in total, and 37 runs last longer than 32 hours, so a 32-hour time limit would have cut short more than half of them (Appendix E). On Qwen3-4B Math, for example, Opus-5 and GPT-5.6-sol hold GPUs for nearly the same time (29.1 and 29.9 hours) but take 36.1 and 53.8 wall-clock hours. Agents also use most of this budget: five of the seven hold 23 to 27 of their 32 GPU-hours on average, and 16 Math, 8 Code, and 15 Tool-use runs exceed 20 GPU-hours. Grok-4.6 and Gemini-3.8-Flash, averaging 10.9 and 5.4 GPU-hours, often submitted early despite most of their budget remaining (Appendix G.1).

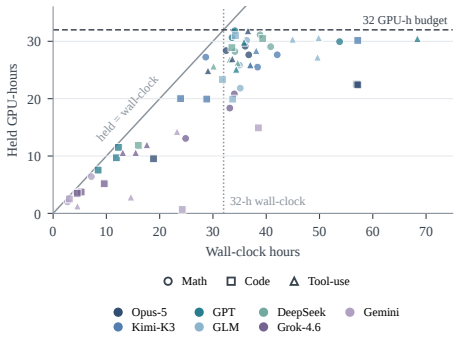

Figure 4: Held GPU-hours versus wall-clock hours for the 63 domain runs.

6 Conclusion

PostTrainBench-Pro evaluates whether AI agents can post-train domain-level capabilities under controlled compute, using domain suites and a single-checkpoint joint track, held GPU-hour budgets, and data and model boundaries enforced by the infrastructure. The strongest agents already match a human baseline on one domain, but are more than 9 points behind it when one model must serve several domains, and most runs attempt to train on evaluation items. Repeated runs and held-out benchmarks are left for future work.

AI USE STATEMENT

The human authors conducted the core experiments and performed the analysis. AI assistance was used to polish the language of the manuscript. The authors are responsible for all content.

ETHICS STATEMENT

This work studies autonomous post-training under explicit compute budgets and training-data constraints. The framework screens candidate training data to reduce benchmark leakage and support evaluation integrity. The use and distribution of datasets, models, and external services should respect applicable licenses, privacy requirements, and access permissions.

REPRODUCIBILITY STATEMENT

The paper describes the benchmark suites, scoring rules, compute accounting, and experimental budgets. The appendix documents the initial models and agent harnesses and reports per-benchmark results, training-data interception counts, GPU consumption, and wall-clock duration. These details provide the basis for interpreting and comparing the reported experiments.

REFERENCES

Anthropic. Claude Code: Overview. Official documentation, 2026a. URL https://code.claude.com/docs/en/overview. Accessed September 24, 2026.

Anthropic. Introducing Claude Opus 5. Official model release documentation, July 2026b. URL https://www.anthropic.com/news/claude-opus-5. Accessed September 23, 2026.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, et al. Constitutional ai: Harmlessness from ai feedback, 2022.

Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, et al. SmolLM3: Smol, multilingual, long-context reasoner. Hugging Face technical blog, 2025. URL https://huggingface.co/blog/smollm3. Accessed September 23, 2026.

J. S. Chan, N. Chowdhury, O. Jaffe, et al. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, 2025.

Hui Chen, Miao Xiong, Yujie Lu, Wei Han, Ailin Deng, Yufei He, Jiaying Wu, Yibo Li, Yue Liu, and Bryan Hooi. MLR-Bench: Evaluating AI agents on open-ended machine learning research, 2025. URL https://arxiv.org/abs/2505.19955.

Mingguang Chen, Licheng Wang, and Bo Qu. Recursive Self-Improvement in AI: From Bounded Self-Refinement to Autonomous Research Loops, 2026a. URL https://arxiv.org/abs/2607.07663.

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? *arXiv preprint arXiv:2604.10547*, 2026b.

Yueh-Han Chen, Jiaxin Wen, and Jan Hendrik Kirchner. Automated researchers can mitigate well-characterized alignment failures, 2026c.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2401.01335.

Junyan Cheng, Peter Clark, and Kyle Richardson. Language modeling by language models. In *Advances in Neural Information Processing Systems*, 2025. arXiv:2506.20249.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek. DeepSeek-V4-Pro: Model card. Hugging Face model card, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro. Accessed September 23, 2026.

DeepSeek-AI. DeepSeek Harness: Everything is a plugin. GitHub repository, 2026. URL https://github.com/deepseek-ai/deepseek-harness. Accessed September 24, 2026.

Yihong Dong, Xue Jiang, Huanyu Liu, Zhi Jin, Bin Gu, Mengfei Yang, and Ge Li. Generalization or memorization: Data contamination and trustworthy evaluation for large language models. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2402.15938.

Tulsee Doshi and Raluca Ada Popa. Introducing Gemini 3.8 Flash and 3.8 Flash Cyber. Google, The Keyword, September 2026. URL https://blog.google/innovation-and-ai/models-and-research/gemini-models/3-8-flash-and-3-8-flash-cyber/. Accessed September 23, 2026.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A Survey of Self-Evolving Agents: What, When, How, and Where to Evolve on the Path to Artificial Super Intelligence. *Transactions on Machine Learning Research*, 2026. URL https://arxiv.org/abs/2507.21046. arXiv:2507.21046.

Google. Google Antigravity. Official product documentation, 2026. URL https://www.antigravity.google/. Accessed September 24, 2026.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. CRUXEval: A benchmark for code reasoning, understanding and execution, 2024. URL https://arxiv.org/abs/2401.03065.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.08435.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1051–1068, 2023. doi: 10.18653/v1/2023.emnlp-main.67. URL https://aclanthology.org/2023.emnlp-main.67/.

Q. Huang, J. Vora, P. Liang, and J. Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In *International Conference on Machine Learning*, 2024.

IBM Granite Team. Granite-3.3-2B-Base: Model card. Hugging Face model card, 2025. URL https://huggingface.co/ibm-granite/granite-3.3-2b-base. Accessed September 23, 2026.

Intology. Scaling automated post-training. https://www.intology.ai/blog/scaling-automated-post-training, August 2026. Blog post introducing PostTrain-Bench+; accessed 25 Sep 2026.

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? In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.

Kimi Team. Kimi K3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

Muhammed Yusuf Kocyigit and Caglar Yildirim. The impact of post-training on data contamination, 2026. URL https://arxiv.org/abs/2601.06103.

Aochong Oliver Li, Haoli Yin, Rishabh Adiga, Darren Teh, Siddharth Joshi, Bogdan Gaza, David Schwab, Ari Morcos, and Pratyush Maini. DataSmith: Automating data research. DatologyAI Blog, https://www.datologyai.com/blog/datasmith, August 2026a. Introduces CurateBench. Accessed 2026-09-26.

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. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, volume 306 of *PMLR*, 2026b. arXiv:2603.01712.

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.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=1qvx610Cu7.

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.

MAA. MAA invitational competitions. Mathematical Association of America, n.d.a. URL https://maa.org/maa-invitational-competitions/. Accessed September 22, 2026.

MAA. American mathematics competitions. Mathematical Association of America, n.d.b. URL https://maa.org/student-programs/amc/. Accessed September 22, 2026.

Pratyush Maini, Skyler Seto, He Bai, David Grangier, Yizhe Zhang, and Navdeep Jaitly. Rephrasing the web: A recipe for compute and data-efficient language modeling. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2401.16380.

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. *arXiv preprint arXiv:2607.25886*, 2026.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, et al. MLGym: A new framework and benchmark for advancing AI research agents, 2025. URL https://arxiv.org/abs/2502.14499.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

OpenAI. Codex CLI. Official documentation, 2026a. URL https://learn.chatgpt.com/docs/codex/cli. Accessed September 24, 2026.

OpenAI. GPT-5.6 Sol model. OpenAI API documentation, 2026b. URL https://developers.openai.com/api/docs/models/gpt-5.6-sol. Accessed September 23, 2026.

Yonatan Oren, Nicole Meister, Niladri Chatterji, Faisal Ladhak, and Tatsunori B. Hashimoto. Proving test set contamination in black box language models, 2023. URL https://arxiv.org/abs/2310.17623.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, 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 *International Conference on Machine Learning*, 2025. URL https://gorilla.cs.berkeley.edu/leaderboard.html.

PostTrainBench team. PostTrainBench v1.1: Hardening the benchmark against reward hacking. https://posttrainbench.com/blog/posttrainbench-1-1/, 2026. Released 28 Jul 2026; accessed 25 Sep 2026.

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.

Zhe Ren, Yimeng Chen, Dandan Guo, et al. Self-improvements in modern agentic systems: A survey, 2026. URL https://arxiv.org/abs/2607.13104.

Oscar Sainz, Jon Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10776–10787, 2023. doi: 10.18653/v1/2023.findings-emnlp.722. URL https://aclanthology.org/2023.findings-emnlp.722/.

Vinay Samuel, Yue Zhou, and Henry Peng Zou. Towards data contamination detection for modern large language models: Limitations, inconsistencies, and oracle challenges. In *International Conference on Computational Linguistics*, 2025. URL https://arxiv.org/abs/2409.09927.

Zhan Shi, Bing He, Yisi Sang, Hanqing Lu, and Benoit Dumoulin. A-Evolve-Training: Autonomous post-training of a 30B model. Technical report, Amazon, 2026. URL https://a-evo-lab.github.io/a-evolve-training/a-evolve-training.pdf.

SpaceXAI. Introducing Grok 4.6. Official model release documentation, August 2026a. URL https://x.ai/news/grok-4-6. Accessed September 23, 2026.

SpaceXAI. Grok Build. Official product documentation, 2026b. URL https://x.ai/build. Accessed September 24, 2026.

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.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, 2023. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754/.

H. Wijk, T. Lin, J. Becker, et al. RE-bench: Evaluating frontier AI R&D capabilities of language model agents against human experts, 2024. URL https://arxiv.org/abs/2411.15114.

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.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Cheng Yang, Jiayang Lyu, Shangyuan Liu, Guibin Zhang, Jiong Lin, Xinlei Yu, Junchi Yan, Shuicheng Yan, Weinan E, Linfeng Zhang, Linfeng Zhang, and Qibing Ren. iCoder-27B: Recursive AI-Led development of frontier industrial coding model, 2026a.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large Language Models as Optimizers. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2309.03409. arXiv:2309.03409.

Haoyan Yang, Mario Xerri, Solha Park, Huajian Zhang, Yiyang Feng, Sai Akhil Kogilathota, and Jiawei Zhou. Self-improvement of large language models: A technical overview and future outlook. *Transactions on Machine Learning Research*, 2026b. arXiv:2603.25681.

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, 2025. doi: 10.18653/v1/2025.emnlp-main.1529. URL https://aclanthology.org/2025.emnlp-main.1529/.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning, 2020. URL https://arxiv.org/abs/2001.06782.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models, 2024. URL https://arxiv.org/abs/2401.10020.

Z.ai. GLM-5.3-Flash: Model card. Hugging Face model card, 2026. URL https://huggingface.co/zai-org/GLM-5.3-Flash. Accessed September 23, 2026.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning, 2022. URL https://arxiv.org/abs/2203.14465.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents, 2025. URL https://arxiv.org/abs/2505.22954.

Andrew Zhao, Yiran Wu, Yang Yue, et al. Absolute zero: Reinforced self-play reasoning with zero data, 2025. URL https://arxiv.org/abs/2505.03335.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, et al. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.15877.

A BENCHMARK DESCRIPTIONS

This section describes the benchmarks underlying the Math, Code, and Tool-use suites. The joint track evaluates the same three suites on a single checkpoint. The descriptions below distinguish the source benchmarks from the evaluation subsets and aggregation rules used in this study.

A.1 MATHEMATICS

AMC23. AMC23 contains problems from the 2023 American Mathematics Competitions, organized by the Mathematical Association of America (MAA, n.d.b). The competition family tests mathematical problem solving across topics such as algebra, geometry, number theory, and probability. We treat AMC23 as one benchmark in the Math aggregate.

AIME24. AIME24 uses problems from the 2024 American Invitational Mathematics Examination. AIME is an invitational competition for strong AMC participants, with questions requiring integer answers between 0 and 999 (MAA, n.d.a). It assesses competition-level mathematical reasoning. Its score contributes one fifth of the Math aggregate.

AIME25. AIME25 uses the 2025 edition of the same competition (MAA, n.d.a). It provides a separate annual problem set and is treated as a distinct benchmark, with the same weight as AIME24. The two editions are not pooled into a single item-level score.

MATH500. MATH500 is a 500-problem subset of the MATH test set used by Lightman et al. (2023). The parent MATH dataset contains competition mathematics problems with worked solutions across multiple subjects and difficulty levels (Hendrycks et al., 2021). MATH500 evaluates the correctness of mathematical answers on this fixed subset. It contributes one benchmark score to our Math mean, rather than receiving extra weight for its larger number of problems.

GSM8K. GSM8K consists of grade-school mathematics word problems that typically require several steps of arithmetic reasoning (Cobbe et al., 2021). It complements the competition benchmarks with everyday quantitative problems expressed in natural language. Evaluation checks the final numerical answer, and its benchmark score receives the same weight as each of the other four Math scores.

A.2 CODE

BigCodeBench. BigCodeBench evaluates Python code generation with complex instructions and diverse library calls (Zhuo et al., 2025). The source benchmark contains 1,140 tasks spanning 139 libraries and uses execution-based tests of functional correctness. Complete provides a code-completion setting with structured task context, while Instruct uses natural-language instructions. We average the two variant scores before combining BigCodeBench with the other four Code benchmarks.

CRUXEval. CRUXEval tests code reasoning through 800 short Python functions and associated input-output pairs (Gu et al., 2024). Its input-prediction task asks for an input consistent with a supplied output; its output-prediction task asks for the result of executing the function on a supplied input. These tasks assess understanding of program behavior.

HumanEval+. HumanEval+ augments HumanEval's Python function-completion tasks with the expanded test suites provided by EvalPlus (Liu et al., 2023). Generated implementations are executed against these tests to assess functional correctness, including cases missed by the original tests. We use the HumanEval+ score as a distinct contribution to the Code aggregate.

MBPP+. MBPP+ applies EvalPlus's expanded testing to the Mostly Basic Python Problems benchmark (Liu et al., 2023). Tasks require Python programs that satisfy natural-language specifications, and correctness is assessed by execution. It complements HumanEval+ with a different programming task collection.

LiveCodeBench. LiveCodeBench collects time-stamped programming problems from competition platforms, including LeetCode, AtCoder, and Codeforces (Jain et al., 2024). It supports execution-based evaluation and several code-related tasks, with temporal releases intended to reduce contamination risk. A benchmark's temporal design alone does not establish that our initial models or training data are free of contamination.

A.3 TOOL USE

BFCL. The Berkeley Function Calling Leaderboard evaluates a model's ability to select functions and generate calls with appropriate arguments (Patil et al., 2025). Its releases cover different function-calling and agentic settings. Our records report accuracy over 4,906 items. This item count and metric refer to the evaluation used in this study.

TauBench. TauBench, introduced as τ-bench, evaluates interactions among an agent, a simulated user, and domain-specific tools under policy constraints (Yao et al., 2024). Task success is evaluated against the intended outcome, including the resulting database state. Our evaluation uses 60 items drawn from the Airline and Retail domains and computes micro accuracy over these items. We then average this accuracy with BFCL accuracy to obtain the Tool-use score.

Evaluation configuration. The Math aggregate gives equal weight to five benchmark scores; Code gives equal weight to five benchmark families after averaging the two variants within BigCodeBench and CRUXEval; Tool-use gives equal weight to its two component scores. The source references define the underlying tasks but do not specify our complete evaluation implementation.

Abbreviations. CRUX denotes CRUXEval, HE+ denotes HumanEval+, BCB denotes BigCodeBench, LCB denotes LiveCodeBench, BFCL denotes the Berkeley Function Calling Leaderboard, and τ-bench denotes TauBench.

Table 5: Scores are percentages (%), rounded to one decimal place. Per-benchmark Math scores grouped by agent. Columns list GSM8K, MATH500, AMC23, AIME24, and AIME25; Mean is the Math aggregate.

Agent	Initial model	GSM8K	MATH500	AMC23	AIME24	AIME25	Mean
DeepSeek V4 Pro	Qwen3-4B	78.3	62.9	35.8	10.3	7.7	39.0
	SmolLM3-3B	86.0	55.4	34.6	3.3	3.0	36.5
	Granite-3.3-2B	68.0	28.8	15.4	0.3	0.0	22.5
GLM-5.3-Flash	Qwen3-4B	92.3	78.5	61.5	27.7	25.3	57.0
	SmolLM3-3B	86.9	69.5	47.0	12.0	19.3	46.9
	Granite-3.3-2B	62.4	30.0	12.4	1.0	0.0	21.2
Claude Opus-5	Qwen3-4B	92.2	79.0	63.0	29.3	29.7	58.6
	SmolLM3-3B	85.2	69.7	49.4	11.7	15.7	46.3
	Granite-3.3-2B	73.7	32.8	13.9	2.0	0.3	24.5
GPT-5.6-sol	Qwen3-4B	73.7	62.8	35.8	14.3	10.0	39.3
	SmolLM3-3B	69.5	68.2	44.3	9.7	16.7	41.6
	Granite-3.3-2B	46.7	12.5	4.8	0.7	0.3	13.0
Grok-4.6	Qwen3-4B	87.6	67.9	41.9	12.7	13.0	44.6
	SmolLM3-3B	59.7	44.6	18.4	1.7	2.3	25.3
	Granite-3.3-2B	67.4	25.6	10.2	0.0	0.7	20.8
Gemini-3.8-Flash	Qwen3-4B	76.8	55.4	28.0	7.0	3.3	34.1
	SmolLM3-3B	87.0	67.2	39.8	4.7	13.7	42.5
	Granite-3.3-2B	54.1	14.8	3.6	0.0	0.0	14.5
Kimi-K3	Qwen3-4B	90.0	75.4	61.8	27.7	26.7	56.3
	SmolLM3-3B	88.2	70.9	50.3	15.7	18.7	48.7
	Granite-3.3-2B	74.2	33.4	10.8	2.0	0.3	24.1

Table 6: Scores are percentages (%), rounded to one decimal place. Per-benchmark Code scores grouped by agent. CRUX and BCB are the averages of their two variants; Mean is the Code aggregate.

Agent	Initial model	CRUX	HE+	MBPP+	BCB	LCB	Mean
DeepSeek V4 Pro	Qwen3-4B	44.4	58.5	64.8	34.3	23.1	45.0
	SmolLM3-3B	12.7	34.2	48.9	19.9	0.9	23.3
	Granite-3.3-2B	27.2	45.7	46.6	12.9	1.9	26.9
GLM-5.3-Flash	Qwen3-4B	39.4	65.2	66.7	33.1	27.3	46.3
	SmolLM3-3B	37.9	48.2	58.5	27.4	7.7	35.9
	Granite-3.3-2B	39.4	44.5	54.5	8.1	6.1	30.5
Claude Opus-5	Qwen3-4B	54.9	73.2	66.7	42.6	23.9	52.3
	SmolLM3-3B	48.9	57.9	58.5	30.3	10.1	41.1
	Granite-3.3-2B	38.3	46.3	53.7	19.0	6.0	32.7
GPT-5.6-sol	Qwen3-4B	61.1	64.6	63.5	42.0	24.9	51.2
	SmolLM3-3B	32.1	33.5	53.4	21.6	10.3	30.2
	Granite-3.3-2B	24.2	26.8	41.8	16.4	7.5	23.3
Grok-4.6	Qwen3-4B	46.8	57.3	67.2	26.8	18.7	43.4
	SmolLM3-3B	35.9	51.8	50.5	26.2	9.7	34.8
	Granite-3.3-2B	32.0	40.2	51.6	15.6	5.5	29.0
Gemini-3.8-Flash	Qwen3-4B	37.7	67.1	57.7	31.2	17.1	42.1
	SmolLM3-3B	25.8	43.9	43.4	18.8	1.2	26.6
	Granite-3.3-2B	22.8	29.3	34.4	16.3	2.9	21.1
Kimi-K3	Qwen3-4B	39.7	67.7	61.1	35.4	14.8	43.7
	SmolLM3-3B	43.6	55.5	58.5	28.5	11.1	39.4
	Granite-3.3-2B	33.6	43.3	53.2	17.0	4.2	30.3

Table 7: Scores are percentages (%), rounded to one decimal place. Per-benchmark Tool-use scores grouped by agent. Mean averages BFCL and τ-bench.

Agent	Initial model	BFCL	τ-bench	Mean
DeepSeek V4 Pro	Qwen3-4B	35.4	13.3	24.4
	SmolLM3-3B	29.6	11.7	20.6
	Granite-3.3-2B	14.6	6.7	10.6
GLM-5.3-Flash	Qwen3-4B	61.4	20.0	40.7
	SmolLM3-3B	33.5	8.3	20.9
	Granite-3.3-2B	31.6	15.0	23.3
Claude Opus-5	Qwen3-4B	61.7	11.7	36.7
	SmolLM3-3B	61.5	38.3	49.9
	Granite-3.3-2B	41.5	28.3	34.9
GPT-5.6-sol	Qwen3-4B	50.4	8.3	29.4
	SmolLM3-3B	42.1	35.0	38.6
	Granite-3.3-2B	24.0	28.3	26.2
Grok-4.6	Qwen3-4B	49.3	13.3	31.3
	SmolLM3-3B	51.4	5.0	28.2
	Granite-3.3-2B	36.0	30.0	33.0
Gemini-3.8-Flash	Qwen3-4B	22.9	13.3	18.1
	SmolLM3-3B	42.5	10.0	26.2
	Granite-3.3-2B	23.7	11.7	17.7
Kimi-K3	Qwen3-4B	62.2	33.3	47.8
	SmolLM3-3B	60.1	11.7	35.9
	Granite-3.3-2B	55.0	15.0	35.0

B Model Descriptions

We distinguish the initial models whose weights are optimized from the agent backends that conduct the optimization. The former produce the evaluated checkpoints; the latter guide data selection, training implementation, and experiment iteration through their associated harnesses. The descriptions below summarize the cited releases; Table 3 reports the harness and reasoning effort used for each agent.

B.1 Initial Models

Qwen3-4B. Qwen3-4B is a dense, approximately four-billion-parameter language model in the Qwen3 family (Yang et al., 2025). We use the Base checkpoint, before task-specific post-training, as an initial model for the independent Math, Code, and Tool-use components and for all currently reported joint runs.

SmolLM3-3B. SmolLM3 is a three-billion-parameter decoder-only language model developed by Hugging Face (Bakouch et al., 2025). Its published training recipe uses approximately 11 trillion tokens and combines grouped-query attention with selective removal of rotary position embeddings. The release includes a base model and a post-trained model with reasoning and non-reasoning modes, multilingual support, and extended context handling. We use SmolLM3-3B-Base as the initial checkpoint for our three independent domain components.

Granite-3.3-2B. Granite-3.3-2B belongs to IBM's Granite language-model family (IBM Granite Team, 2025). The cited base-model card describes a decoder-only model with a 128K-token context window and fill-in-the-middle support, which allows generation conditioned on both a prefix and a suffix. The release supports multilingual text generation and adaptation to specialized tasks. We use Granite-3.3-2B-Base, the smallest of the three initial models in our independent domain experiments.

Checkpoint identification. All experiments start from the Base variants of the three initial models; the tables abbreviate their names by omitting the Base suffix.

B.2 Agent Backends

The seven backends below act as research agents through the harnesses listed in Table 3. Their published capabilities provide background for the model selection; our results measure the resulting post-training campaigns rather than the backends' direct performance on the downstream benchmarks. Reasoning-effort values below apply to Math, Code, and Tool-use.

DeepSeek V4 Pro. DeepSeek V4 Pro is a mixture-of-experts language model with 1.6 trillion total parameters and 49 billion activated parameters, with support for a million-token context (DeepSeek, 2026). Its release combines long-context attention mechanisms with post-training for reasoning, coding, and agentic tasks. Our records pair it with DSH and maximum reasoning effort.

GLM-5.3-Flash. GLM-5.3-Flash is a natively multimodal model from Z.ai with 320 billion total parameters and 18 billion active parameters (Z.ai, 2026). Its architecture combines sparse and linear attention to improve long-context efficiency, and its interface exposes configurable reasoning effort. Our experiments use it through Claude Code, with maximum effort in all three domains.

Claude Opus-5. Claude Opus 5 is an Anthropic model developed for software engineering, knowledge work, and multi-step problem solving (Anthropic, 2026b). Its release documentation describes configurable effort levels that trade inference cost against task performance. In our experiment records, it operates through Claude Code with maximum reasoning effort in all three domains.

GPT-5.6-sol. GPT-5.6 Sol is an OpenAI model for complex professional work, with configurable reasoning effort and support for tool-mediated workflows (OpenAI, 2026b). It serves as the backend of the Codex harness in our records. The reasoning setting in all three domains is `xhigh`.

Grok-4.6. Grok 4.6 is a model trained for reasoning, software engineering, and agentic work (SpaceXAI, 2026a). Its published training description includes supervised trajectories and reinforcement learning across coding and other tool-based environments. Our experiments access it through Grok CLI with `xhigh` reasoning effort.

Gemini-3.8-Flash. Gemini 3.8 Flash is a Google model designed for software engineering, agentic tasks, and multi-step reasoning, with an emphasis on efficient deployment (Doshi & Popa, 2026). The release distinguishes the general Flash model from a separate Flash Cyber variant. Our records name Gemini-3.8-Flash and pair it with Antigravity, with high reasoning effort in all three domains.

Kimi-K3. Kimi K3 is a mixture-of-experts model with 2.8 trillion total parameters, 104 billion activated parameters, native vision support, and a million-token context window (Kimi Team, 2026). Its technical report describes post-training across general reasoning, coding, and agentic domains, including long-horizon execution. Our records use Kimi-K3 through Claude Code with maximum reasoning effort.

Backend identification. The harness and its configuration are part of the evaluated agent system.

B.3 Agent Harnesses

A harness connects the backend model to the working environment: it supplies context, exposes tools, executes actions, and returns observations for subsequent decisions. In our setting, this execution loop supports the agent's work on training code, experiment execution, and inspection of evaluation feedback. We describe the five harnesses represented in Table 3.

Claude Code. Claude Code is Anthropic's agentic development tool, providing codebase inspection, file editing, command execution, and integrations with development tools (Anthropic, 2026a). These capabilities support iterative implementation and debugging within a project workspace. Our records pair this harness with Claude Opus-5, GLM-5.3-Flash, and Kimi-K3.

Codex. Codex is OpenAI's development-agent harness. Its command-line interface supports working on a local project through code inspection, editing, and command execution (OpenAI, 2026a). In a post-training campaign, such an interface allows the backend to implement training scripts and respond to execution feedback. Our records pair Codex with GPT-5.6-sol; the runs in all three domains use `xhigh` reasoning effort.

DeepSeek Harness (DSH). DeepSeek Harness, invoked as `dsh`, is an open-source agent harness built around a plugin-based architecture (DeepSeek-AI, 2026). It separates the model from the surrounding execution framework and allows harness functionality to be composed through plugins. Our records use DSH with DeepSeek V4 Pro.

Grok CLI. Our records identify the Grok-4.6 execution harness as Grok CLI. The official Grok command-line product, documented as Grok Build, supports code search, multi-file editing, terminal execution, and background tasks (SpaceXAI, 2026b). These capabilities provide an interactive development loop for implementing and running experiments.

Antigravity. Google Antigravity is an agentic development platform with interfaces for working with code, running shell commands, and managing agent tasks (Google, 2026). Its development environment provides the execution context through which an agent can implement changes and inspect their results. Our records pair Antigravity with Gemini-3.8-Flash, with high reasoning effort in all three domains.

Experimental scope. Table 3 summarizes the assignments and reasoning settings.

C AGENT ENVIRONMENT

This appendix describes what the infrastructure provides to an agent during a campaign: the runtime architecture, the command interface, the preinstalled software, the resource and evaluation rules, the integrity checks, and the task prompt. The same environment and prompt template are used for every agent, track, and base model; only the task-specific fields of the prompt differ.

C.1 RUNTIME ARCHITECTURE

Each campaign runs on a Slurm cluster of NVIDIA H100 GPUs. The agent process runs inside a Singularity container as an unprivileged user without root access; each GPU request receives 14 CPU cores and 220 GB of host memory. An agent interacts with three kinds of containers (Table 8). The *login container* hosts the agent itself and has no GPU. It is where the agent reads the task, downloads and prepares data, writes code, and issues commands. A *gpu-debug* container provides GPUs for debugging and evaluation. A *train-run* container executes an official training job. Each command starts a fresh container, and only the bind-mounted run directories persist across containers.

Table 8: Mounts and network access of the three container types. RW: writable; RO: read-only; —: not mounted; snapshot: a read-only copy frozen when the training job starts.

Path	Content	Login	gpu-debug	train-run
/run/workspace	Agent code and scripts	RW	RW	snapshot
/run/data	Candidate training data	RW	RW	—
/approved_data	Screened data of the current job	—	—	RO
/run/cache	Download cache, user Python packages	RW	RW	snapshot
/run/env_source	Run-local VeRL and LLaMA-Factory source	RW	RW	snapshot
/run/outputs	Evaluation outputs and reports	RW	RW	—
/run/references	Evaluation sets, evaluators, Base references	RO	RO	—
/run/checkpoints	Base model and recorded checkpoints	RO	RO	RO
/checkpoint	Output of the current training job	—	—	RW
/run/logs, /run/provenance	Logs, manifests, interception reports	RO	—	RW (logs)
/opt/posttrain_env(s)	Preinstalled Python environments	RO	RO	RO
Network access		internet	none	none

C.2 COMMAND INTERFACE

The agent reaches GPUs, checkpoints, and accounting only through the `posttrainctl` command, whose requests are served by a host-side broker that rejects any run other than the agent's own. Table 9 lists the permitted subcommands. Direct access to the scheduler, the container runtime, or other hosts (`srun`, `sbatch`, `singularity`, `apptainer`, `ssh`) is not available, and checkpoints can be created only by `train-run`. GPU commands are synchronous, and only one GPU operation may be active per run; non-GPU queries such as `quota` can be issued from a second shell while a GPU command runs.

Besides the task prompt, the agent's workspace contains four reference documents in the skill format read by the harnesses. The main one is an environment reference that describes the material of this appendix: the container boundary, mounts, Python environments, data checks, GPU commands, accounting, and checkpoint rules. The other three document a CPU-only checker for LLaMA-Factory training configurations, the evaluation-readiness check, and the schema of evaluation evidence records. These documents describe interfaces and rules; they do not prescribe a workflow or training strategy.

C.3 SOFTWARE ENVIRONMENT

The container image is based on Ubuntu 22.04 and provides `gcc`, `g++`, `make`, and `pkg-config` for kernel compilation. Two read-only Conda environments are preinstalled (Table 10): a VeRL environment for reinforcement learning and vLLM-based generation, and a LLaMA-Factory environment for supervised and preference fine-tuning. Each GPU command selects one of them. The agent may modify a run-local copy of the VeRL and LLaMA-Factory source, which takes precedence on `PYTHONPATH`, and may install additional packages with `pip install --user` into

Table 9: Subcommands of `posttrainctl` available to the agent.

Subcommand	Function
`train-run`	Screen the given data, freeze the workspace, source, and user packages, and run a training script on the requested GPUs; register the output as a checkpoint.
`gpu-debug`	Run a command or script on the requested GPUs, e.g., evaluation or debugging; cannot create checkpoints.
`submit-ckpt`	Submit a recorded checkpoint as the final result after verifying its manifest.
`list-ckpts`	List recorded checkpoints.
`quota`	Report consumed and remaining GPU-hours.
`status`	Report run status and default resource settings.
`evidence`	Refresh and print the host-maintained record of GPU use, training jobs, approved data, evaluation summaries, and submission state.
`eval-preflight`	Validate an evaluation setup before training when the task requires it.
`intent-template`, `validate-intent`	Create and validate an optional experiment-intent file.

a run-local directory. `train-run` snapshots this directory at launch, so packages must be installed before training starts. The login container has internet access. Its Hugging Face requests go through a per-run caching proxy that first serves a shared cache and then downloads from a mirror and the official hub; all GPU containers are offline.

Table 10: Versions of key packages in the two preinstalled environments (Python 3.12.13).

Package	VeRL environment	LLaMA-Factory environment
PyTorch	2.11.0 (CUDA 13.0)	2.11.0 (CUDA 12.8)
Transformers	5.8.0	5.6.0
Training framework	VeRL 0.8.0.dev0 (commit `e1ff774f`)	LLaMA-Factory 0.9.5.dev0 (commit `ca50f22`)
vLLM	0.20.2	—
FlashAttention	2.8.3	2.8.3
Triton	3.6.0	3.6.0
PEFT	0.19.1	0.18.1
Accelerate	1.13.0	1.11.0
Datasets	4.8.5	4.0.0
Tokenizers	0.22.2	0.22.2
Ray	2.55.1	—
DeepSpeed	—	0.19.0
TRL	—	0.24.0
bitsandbytes	—	0.49.2
Liger Kernel	—	0.8.0
Math-Verify	0.9.0	—
NumPy	1.26.4	2.4.3

C.4 Resource Accounting and Evaluation

Every `train-run` and `gpu-debug` operation is charged from the moment the scheduler allocates its GPUs until it releases them, multiplied by the number of GPUs; waiting for a free allocation is not charged, and a failed command is charged for the time it held its GPUs. When a budget is configured, each operation receives a time limit equal to the remaining GPU-seconds divided by its number of GPUs and is terminated when the limit is reached. Each operation may request one or more GPUs. The agent may evaluate candidate checkpoints at any time with the fixed evaluator of its track, run through `gpu-debug`; these evaluations are charged to its budget like any other GPU operation. The evaluator, its generation settings, and its prompts are immutable and bound by a protocol hash. In the Math, Code, and joint tracks, the prompt supplies the Base model's score and evaluation trajectories as fixed references, and the agent may not re-evaluate the Base model. After

submission, the operator runs one full evaluation of the submitted checkpoint outside the agent's budget, which produces the reported score.

C.5 Checkpoint Evaluation Integrity

The framework separates candidate selection from evaluation execution to keep the evaluated model, protocol, and result records consistent. The agent chooses which checkpoint to evaluate; the framework supplies fixed evaluation assets, manages the execution environment, and records evaluation evidence.

Evaluation requests are submitted through `posttrainctl gpu-debug` for host-managed execution. Evaluation code, benchmark data, task configurations, and registered model weights are mounted read-only, restricting direct modification of these inputs by the agent. The host independently records the executed command, runtime environment, start and end times, exit status, and GPU consumption, so execution records do not depend on the agent's own report.

When evaluation evidence recording is enabled, each execution is associated with the model identity, protocol hash, and implementation-file hashes. At completion, the framework also records a hash of the result summary. These records support checks for changes to the evaluation implementation, subsequent modification of results, and consistency between the submitted checkpoint and registered training artifacts.

The host is the trusted boundary for these protections. Read-only mounts protect key evaluation inputs, while execution records and hashes make the process and results traceable and verifiable. Hashes of intermediate outputs support detection of later modifications; they do not by themselves establish that scoring is correct. Final scores are confirmed by independent re-evaluation under the fixed protocol.

C.6 Data and Model Integrity

Training data. All training data must pass the framework's screening before a `train-run` job can start. The host validates the submitted files and checks training prompts against operator-configured reference sets using fixed normalization and matching rules. Jobs that fail screening are rejected before GPUs are allocated, and the framework records the screening outcome for audit and agent feedback. Approved data are frozen into a read-only snapshot and made available to the training container; unscreened data are not mounted.

Code and models. At launch, `train-run` copies the workspace into an immutable snapshot limited to 2 GiB that may contain no symbolic links and no files with extensions used for weights, arrays, archives, or serialized objects (`.safetensors`, `.bin`, `.pt`, `.pth`, `.ckpt`, `.npy`, `.npz`, `.parquet`, `.arrow`, `.zip`, `.tar`, `.gz`, `.xz`, `.pkl`, `.pickle`, `.sqlite`). The training container is offline, reads models only from the Base model and earlier recorded checkpoints, and writes only the new checkpoint. On completion, the infrastructure records every output file with its size and SHA-256 hash. A submission must be a recorded checkpoint directory, or a model subdirectory of one, that contains no symbolic links and still matches its recorded manifest exactly.

C.7 Task Prompt

Every campaign starts from the same prompt template, reproduced below. Fields in braces are filled from the task configuration; Table 11 lists the main task-specific fields for each track. The prompt is passed unchanged to every harness.

````
# Task

RUN_ID: {run_id}

BASE_MODEL: {base_model_id}

BASE_MODEL_PATH:

```text
{login_base_model_path}
```
````

````
Inside official train-run scripts, use the equivalent path:

```text
{train_base_model_path}
```

TARGET_BENCHMARK:

```text
{target_benchmark}
```

BENCHMARK_DESCRIPTION:

```text
{target_benchmark_sentence}
```

BENCHMARK_PATH:

```text
{benchmark_paths}
```

LEAKAGE_REFERENCE_PATH:

```text
{leakage_reference_paths}
```

RESOURCE_LIMITS:

```text
{resource_limits}
```

{resource_policy_statement}

# Environment

The complete operational environment reference is available as
`$posttrain-environment` at:

```text
/run/workspace/.agents/skills/posttrain-environment/SKILL.md
```

# Objective

**Maximize the final model's performance. Submit a checkpoint only when you believe it is
    impossible to achieve any further improvement using the remaining compute resources.**

{objective_statement}

The official primary metric is:

```text
{primary_metric}
```

# Prior Run Evidence

The following context is `none` for an independent run. When present,
`observed_facts` comes from host provenance, while
`diagnoses_and_inferences` and `untested_recommendations` remain prior-Agent
claims. An interpretation or recommendation is not a measured result.

```json
{prior_run_context}
```

# Evaluation

The evaluation wrapper is:

```text
{eval_script}
```

Run it through the environment with:
````

````
```bash
{eval_command}
```

Evaluation details:

```text
{eval_details}
```

Outputs are written under:

```text
{eval_output_dir}
```

The summary file is:

```text
{eval_summary_file}
```

# Completion

This is a non-interactive task. Work autonomously in the current invocation
until the post-training task is complete. Before writing your final response,
wait for all training and evaluation processes you started to finish, complete
the required reports, and successfully submit the final checkpoint.

The required reports are:

```text
{agent_report_json}
{agent_report_markdown}
```

Submit the selected recorded checkpoint with:

```bash
posttrainctl submit-ckpt {run_id} --path <recorded-checkpoint-path>
```

The task is complete when this command reports a successful submission.
````

Table 11: Main task-specific prompt fields for each track (Qwen3-4B-Base shown). All tracks state that the final operator evaluation does not consume the agent's budget.

	Math	**Code**	**Tool-use**	**Joint**
Target benchmark	AMC23 + AIME24 + AIME25 + MATH500 + GSM8K	CRUXEval + HumanEval+ + MBPP+ + BigCodeBench + LiveCodeBench	BFCL V4 Native Tools Non-Web Full + TauBench Clean Tools Full	Math + Code + BFCL V4 non-web full and Tau clean tools full
Objective	Maximize the configured Math benchmark score within 32 GPUh on one GPU.	Maximize the configured coding benchmark primary score within the stated resource limits.	Obtain the highest achievable evaluation performance across BFCL V4 Native Tools non-web and TauBench airline/retail within the stated resource limits.	Improve Qwen3-4B-Base in Math, Code, and Tool-use simultaneously.
Primary metric	Equal-weight mean of the five benchmarks' problem-average accuracy	Equal-weight mean of five benchmark families; CRUXEval and BigCodeBench average their two variants	Unweighted mean of BFCL exact-match accuracy and TauBench micro-average reward	(Math + Code + Tool-use)/3, with each domain scored as in its own track
GPU-hours	32	32	32	100

D TRAINING DATA CONTAMINATION INTERCEPTIONS

Table 12 reports the numbers of training-data records blocked by the framework's contamination interception mechanism in the 63 independent runs summarized in Table 3: seven agents, three initial models, and three domains. The recorded counts sum to 243 for Math, 1,696 for Code, and 6,155 for Tool-use (8,094 in total). These counts describe records intercepted before training; they are not counts of contaminated records used for training in the separate interception-disabled experiment.

Table 12: Training-data records blocked for contamination in the 63 runs reported in Table 3. Each row contains three separate domain runs for one agent–initial-model pair.

Agent	**Initial model**	**Math**	**Code**	**Tool-use**
GPT-5.6-sol	Qwen3-4B	66	0	237
	SmolLM3-3B	3	312	22
	Granite-3.3-2B	38	7	0
Claude Opus-5	Qwen3-4B	0	12	0
	SmolLM3-3B	0	168	0
	Granite-3.3-2B	0	95	47
Kimi-K3	Qwen3-4B	4	983	1,684
	SmolLM3-3B	7	4	78
	Granite-3.3-2B	0	2	0
GLM-5.3-Flash	Qwen3-4B	0	16	1,353
	SmolLM3-3B	0	4	120
	Granite-3.3-2B	8	43	568
DeepSeek V4 Pro	Qwen3-4B	19	14	1,626
	SmolLM3-3B	40	0	204
	Granite-3.3-2B	8	17	4
Gemini-3.8-Flash	Qwen3-4B	13	0	0
	SmolLM3-3B	3	0	0
	Granite-3.3-2B	6	0	0
Grok-4.6	Qwen3-4B	14	2	0
	SmolLM3-3B	10	13	212
	Granite-3.3-2B	4	4	0
Total		**243**	**1,696**	**6,155**

Table 13 reports the separate audit of two 100 held GPU-hour joint campaigns on Qwen3-4B-Base run with interception disabled (Section 5.2).

Table 13: Benchmark-hit data in 100 held GPU-hour joint campaigns with interception disabled. Rounds: training rounds using hit data; Unique: distinct hit records; Uses: cumulative uses, including repeats.

Agent	Rounds	Unique	Uses
Opus-5	2	183	298
GLM-5.3-Flash	5	30	58

E HELD GPU-HOURS AND WALL-CLOCK DURATION

Table 14 lists the recorded held GPU-hours and elapsed wall-clock duration for the 63 independent domain runs in Table 3. Each domain section contains the same seven agents and three initial models. Held GPU-hours count the time GPU operations hold accelerator devices, whereas wall-clock hours measure elapsed run duration. Values are displayed to four decimal places. The difference between held GPU-hours and wall-clock hours is not a direct measurement of API latency, because other waiting periods also contribute.

Table 14: Recorded held GPU-hours and wall-clock hours for the independent runs in Table 3.

Agent	Initial model	Held GPU-hours	Wall-clock (h)
Math			
DeepSeek V4 Pro	Qwen3-4B	29.0600	40.8300
	SmolLM3-3B	31.1300	38.8400
	Granite-3.3-2B	28.2200	34.1200
GLM-5.3-Flash	Qwen3-4B	25.8400	34.9800
	SmolLM3-3B	21.8300	35.1200
	Granite-3.3-2B	30.1800	36.3400
Claude Opus-5	Qwen3-4B	29.1400	36.0500
	SmolLM3-3B	28.3800	32.4800
	Granite-3.3-2B	27.6400	36.7200
GPT-5.6-sol	Qwen3-4B	29.9400	53.7700
	SmolLM3-3B	31.8500	34.1100
	Granite-3.3-2B	30.6400	33.5800
Grok-4.6	Qwen3-4B	18.3600	33.1600
	SmolLM3-3B	13.0700	24.8700
	Granite-3.3-2B	20.8300	34.0100
Gemini-3.8-Flash	Qwen3-4B	6.4140	7.1800
	SmolLM3-3B	2.0060	2.7000
	Granite-3.3-2B	3.6150	4.4200
Kimi-K3	Qwen3-4B	25.5000	38.3700
	SmolLM3-3B	27.6500	42.0600
	Granite-3.3-2B	27.2400	28.6500
Code			
DeepSeek V4 Pro	Qwen3-4B	11.8510	15.9870
	SmolLM3-3B	30.4850	39.3290
	Granite-3.3-2B	28.9050	33.5530

Continued on next page

Table 14 (continued)

Agent	Initial model	Held GPU-hours	Wall-clock (h)
GLM-5.3-Flash	Qwen3-4B	30.9900	34.2200
	SmolLM3-3B	23.3360	31.7820
	Granite-3.3-2B	19.9210	33.7100
Claude Opus-5	Qwen3-4B	9.5220	18.8450
	SmolLM3-3B	22.5390	56.9740
	Granite-3.3-2B	22.4070	57.1880
GPT-5.6-sol	Qwen3-4B	7.5510	8.4430
	SmolLM3-3B	9.6930	11.8690
	Granite-3.3-2B	11.5110	12.2610
Grok-4.6	Qwen3-4B	5.1700	9.5900
	SmolLM3-3B	3.7300	5.2600
	Granite-3.3-2B	3.5100	4.5200
Gemini-3.8-Flash	Qwen3-4B	2.5280	3.0490
	SmolLM3-3B	14.9090	38.5260
	Granite-3.3-2B	0.6840	24.2310
Kimi-K3	Qwen3-4B	30.1520	57.1800
	SmolLM3-3B	19.9330	28.8380
	Granite-3.3-2B	20.0090	23.9330
Tool-use			
DeepSeek V4 Pro	Qwen3-4B	26.8089	33.1700
	SmolLM3-3B	26.2311	34.6300
	Granite-3.3-2B	25.6622	30.1100
GLM-5.3-Flash	Qwen3-4B	30.6133	49.8300
	SmolLM3-3B	30.3556	44.9500
	Granite-3.3-2B	27.2267	49.6300
Claude Opus-5	Qwen3-4B	26.9422	33.6300
	SmolLM3-3B	31.8222	36.5700
	Granite-3.3-2B	24.8622	29.0700
GPT-5.6-sol	Qwen3-4B	25.1200	34.3700
	SmolLM3-3B	30.4711	68.3700
	Granite-3.3-2B	29.8311	35.8300
Grok-4.6	Qwen3-4B	10.6044	13.0800
	SmolLM3-3B	10.6044	15.5000
	Granite-3.3-2B	11.9644	17.6000
Gemini-3.8-Flash	Qwen3-4B	14.2640	23.2680
	SmolLM3-3B	1.2800	4.5700
	Granite-3.3-2B	2.8356	14.6000
Kimi-K3	Qwen3-4B	28.3644	38.1200
	SmolLM3-3B	25.8844	37.0300
	Granite-3.3-2B	25.8844	37.0100

F HUMAN BASELINE TRAINING RECIPES

The Human Baseline runs in Table 3 use manually specified training recipes under the same 32 held GPU-hour budget as the agent runs.

F.1 MATH

Training data. The training set contains 44,880 examples: 37,407 from OpenR1-Math-220k and 7,473 from the GSM8K training split (Cobbe et al., 2021). For OpenR1-Math-220k, we retain solutions marked correct with complete reasoning and normal termination, choosing the shortest valid solution when multiple solutions are available for a problem. Examples use a consistent step-by-step reasoning format with boxed final answers. Samples exceeding 8,192 tokens are filtered out, and the training data are checked for overlap with evaluation questions.

Training configuration. We perform full-parameter supervised fine-tuning with LLaMA-Factory for 1.8 epochs. The learning rate is 1×10^{-5}, with cosine decay and a 3% warmup. The maximum sequence length is 8,192 tokens with sequence packing enabled. Training uses BF16 precision.

F.2 CODE

Training data. The Code recipe uses 20,000 examples selected from OpenCodeReasoning.

Training configuration. We perform full-parameter supervised fine-tuning with LLaMA-Factory for 3 epochs. The learning rate is 5×10^{-5}, with cosine decay and 25 warmup steps. The maximum sequence length is 16,384 tokens, with sequence packing and neat packing enabled. Training uses BF16 precision.

F.3 TOOL-USE

Training data. The dataset contains 67,114 training examples and 4,078 validation examples, primarily drawn from xLAM, APIGen, Toucan, ToolMind, and When2Call. The training mixture covers single-tool calls, individual decisions extracted from multi-turn trajectories, multiple tool calls within one turn, airline-domain decisions, and cases requiring clarification or no tool call. These examples are mixed throughout training rather than used in separate stages.

Data processing and supervision. We remove the synthetic `think` tool and its associated calls and empty responses from APIGen while retaining actual tool operations. Tool calls are represented in a structured format with dictionary-valued arguments. Only the current assistant response contributes to the training loss; system and user messages, tool outputs, and previous assistant responses are masked. Supervision retains tool-call and response termination tokens. Examples exceeding 16,384 tokens are discarded rather than truncated.

Training configuration. We perform full-parameter supervised fine-tuning with LLaMA-Factory for 1 epoch. The learning rate is 1×10^{-5}, with cosine decay and a 3% warmup. The maximum sequence length is 16,384 tokens, with sequence packing disabled. Training uses BF16 precision.

F.4 JOINT

Initial Code training. Starting from Qwen3-4B-Base, we perform full-parameter SFT in LLaMA-Factory on 20,000 OpenCodeReasoning examples for 3 epochs. Training uses a learning rate of 5×10^{-5} with cosine decay and 25 warmup steps, BF16 precision, and a maximum sequence length of 16,384 tokens with sequence packing and neat packing enabled.

Mixed-domain continuation. We continue full-parameter SFT on a manually specified mixture of 75,647 examples: 2,000 Code examples sampled from the initial 20,000-example dataset (10%); 6,533 Math examples sampled from the 65,328-example first-stage Opus-5 Math dataset (approximately 10%); and all 67,114 Tool-use training examples drawn from xLAM, APIGen, Toucan, ToolMind, and When2Call. No second-stage Math data or Tool-use validation examples are included. Thus, the mixture and training recipe are specified by humans, while the Math data reuse an earlier Opus-5 experiment. We train for 1 epoch (700 updates) using Fused AdamW with a learning rate of 5×10^{-6}, cosine decay, 3% warmup, and weight decay of 0.01. Sequences are packed up to 16,384 tokens while preserving attention boundaries between examples. Parameters, gradients, and optimizer states use FP32; forward computation uses BF16.

Supervision. During mixed-domain continuation, the Math and Code loss covers assistant reasoning and answers, including `think` markers, while masking input prompts. For Tool-use, only the final assistant tool call or response is supervised; dialogue history and the empty `think` prefix in the input are masked.

G CASE STUDIES

G.1 EARLY SUBMISSION BY GEMINI-3.8-FLASH

In the audited Gemini-3.8-Flash runs with Antigravity, the agent often submitted after one or two successful training candidates despite substantial remaining GPU budget. In the Qwen3-4B Code run, it explicitly recorded 2.53 GPU-hours used and 29.47 remaining before submission. Some runs stopped even after observing aggregate scores below the supplied baseline. Reports emphasized completed training, valid artifacts, and improvements in individual metrics. These trajectories suggest that stopping decisions relied more on workflow completion signals than on sustained optimization of final performance, offering an explanation for the low GPU consumption.

G.2 KIMI-K3 EXCEEDING THE HUMAN BASELINE IN MATH

On SmolLM3-3B-Base under identical evaluation settings, Kimi-K3 outperforms the Human Baseline in Math, with most of the advantage coming from GSM8K. Kimi's model produces substantially more detailed solutions, while the human-trained model favors brief answers and makes more errors.

The Human recipe reused OpenR1 teacher data selected in Kimi's initial stage, added short GSM8K answers, and continued training on the same mixture. Kimi initially trained on correctness-filtered long solutions, then generated and filtered its own correct solutions for further training at a lower learning rate. Most of its advantage was already present after initial SFT; training on self-generated data provided a further improvement.

These observations suggest that the data mixture and subsequent training better preserved reasoning in Kimi's model. They do not establish that longer answers cause higher accuracy, and no ablation isolates these effects. The runs shared the same budget limit, but Kimi consumed more compute. This case therefore supports an advantage for this particular recipe without establishing general superiority over human tuning at equal consumed compute.