M1: TOWARDS A GENERALIST AGENTIC MODEL FOR AUTONOMOUS MACHINE LEARNING ENGINEERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are expanding from assisting scientific research and engineering to autonomously developing and improving AI systems, advancing AI for AI (AI4AI) as a pathway toward recursive self-improvement (RSI). Workflow-driven machine learning engineering (MLE) systems provide a practical path toward this pursuit through complex orchestration that manages long-horizon experimentation, while recent MLE systems adopt model-directed interaction, allowing models to autonomously organize experiments and adapt optimization strategies based on execution feedback. However, feedback from each experiment reflects several preceding decisions, and a long MLE run contains many such costly experiments. Training must therefore select useful supervision from long interaction trajectories and learn from intermediate outcomes as they become available. To address these challenges, we propose a post-training method for autonomous long-horizon MLE optimization and apply it to train M1, a 35B Mixture-of-Experts agentic model. We design the Kaggle-MLE Factory to construct executable tasks and an interaction harness for model-directed experimentation. Hierarchical agentic SFT transfers general tool-use and coding capabilities to MLE through outcome-guided supervision. Windowed Interaction RL (WIRL) treats the multi-turn decisions between successive experiment results as an execution window and assigns it a shared outcome-based reward. Asynchronous collection and policy correction support policy updates before the trajectory ends. On MLE-Bench Lite, M1 improves the average medal rate of the base model from 37.9% to 72.7%, achieving performance comparable to larger frontier models such as Kimi-K3 and GLM-5.3. Further experiments show that M1 improves performance on NatureBench Lite, agent scaffold optimization, and algorithm optimization, suggesting that the learned optimization capability generalizes to other tasks.

1 INTRODUCTION

The rapid advancement of reasoning, code generation, and tool interaction in large language models (LLMs) is enabling artificial intelligence (AI) to move beyond assisting scientific research and engineering toward autonomously developing and improving AI systems. These efforts include model training, optimization of solver programs, and algorithm design, making AI for AI (AI4AI) a research direction toward recursive self-improvement (RSI). In this context, machine learning engineering (MLE) provides a concrete setting for this goal, requiring long-horizon optimization through repeated execution, feedback analysis, and revision rather than one-shot generation.

Most existing MLE systems (Jiang et al., 2025; Nam et al., 2025; Yang et al., 2025) extend code generation into iterative optimization through search, memory, or multi-agent collaboration. As shown in Figure 1(a), predefined workflows or search rules typically control experiment sequencing and stage transitions, while LLMs primarily generate or debug code at designated steps. These systems rely heavily on manually designed control logic to organize long-horizon experimentation.

Recently, Arbor (Jin et al., 2026) and Argus (Li et al., 2026a) maintain persistent research states and allow models to design, continue, and revise experiments, placing greater demands on the model's ability to plan experiments and sustain optimization over long interaction horizons. ML-Agent (Liu et al., 2025) combines multi-turn demonstrations with single-step RL, while Frontis-MA1 (Yang et al., 2026a) trains program evolution operators through SFT and RL and integrates them into an

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

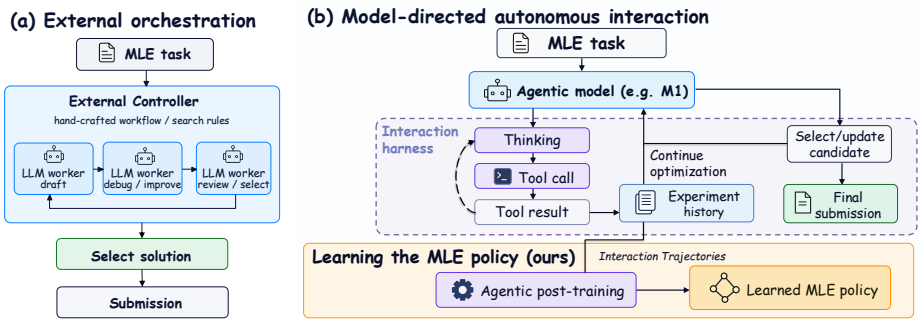


Figure 1: Decision making in long-horizon MLE optimization. (a) External orchestration organizes optimization through predefined workflows and search rules. (b) In model-directed interaction, the model selects subsequent actions based on experimental states and execution feedback.

external evolutionary search framework. These approaches primarily train models for single-step decisions or local operations within externally organized processes. In contrast, we study how to train models to organize experiments and improve solutions over sustained interaction, using experience and execution feedback from real-world MLE tasks. However, training this long-horizon optimization capability poses three challenges. 1) Open-ended MLE requires an interaction environment that preserves experimental states while leaving action selection to the model. 2) High-quality long-horizon supervision is difficult to obtain because MLE tasks are heterogeneous and interaction trajectories may mix useful exploration and failure recovery with degraded decisions. 3) Each costly experiment returns feedback on several preceding decisions before the full run ends; online RL must use this intermediate signal while handling policy mismatch during asynchronous collection.

To address these challenges, we propose a post-training method for autonomous MLE that combines a model-directed long-horizon interaction harness, hierarchical agentic SFT, and Windowed Interaction Reinforcement Learning (WIRL). Based on this, we train **M1**, a 35B Mixture-of-Experts (MoE) agentic model, to learn long-horizon optimization strategies. As shown in Figure 1(b), M1 uses sustained tool interaction within the interaction harness to autonomously organize experiments, revise solutions based on execution feedback, and maintain submission candidates. We build the Kaggle-MLE Factory to convert historical Kaggle competitions into executable MLE tasks. Hierarchical agentic SFT transfers general tool-use and coding capabilities to MLE optimization, with execution outcomes guiding the selection of trajectories and segments for supervision. To use this intermediate feedback, WIRL treats each completed experiment as the boundary of an execution window containing decisions since the previous result. It trains those decisions with a shared outcome-based reward, while asynchronous collection and policy correction allow updates as trajectories continue. On MLE-Bench Lite, M1 improves the average medal rate of the base model from 37.9% to 72.7%, achieving performance comparable to larger models such as Kimi-K3 and GLM-5.3. M1 also improves performance on NatureBench Lite and transfers its optimization capabilities to agent scaffold and algorithm optimization tasks. Our key contributions are:

- We introduce **M1**, a 35B MoE model for autonomous MLE, trained with hierarchical agentic SFT and WIRL. WIRL links execution feedback to multi-turn decisions within execution windows and combines local progress rewards with policy correction for asynchronous online learning.
- We build the **Kaggle-MLE Factory** and an **Interaction Harness** to provide executable tasks and model-directed long-horizon interaction for training and evaluation.
- Extensive experiments show that M1 improves the base model's average medal rate from 37.9% to 72.7% on MLE-Bench Lite, improves performance on NatureBench Lite, and generalizes to agent scaffold and algorithm optimization tasks.

2 PRELIMINARY

We formulate MLE as the iterative optimization of executable ML solutions within a given time budget B_e. An MLE task is represented as $e = (d_e, \mathcal{D}_e, m_e)$, where d_e is the task description, $\mathcal{D}_e$ is the task data, and m_e is the evaluation metric. A candidate solution $y \in \mathcal{Y}_e$ represents a complete ML pipeline including data processing, feature engineering, model training, and prediction, where $\mathcal{Y}_e$ denotes the solution space. The objective is to find a best-performing solution for a given task,

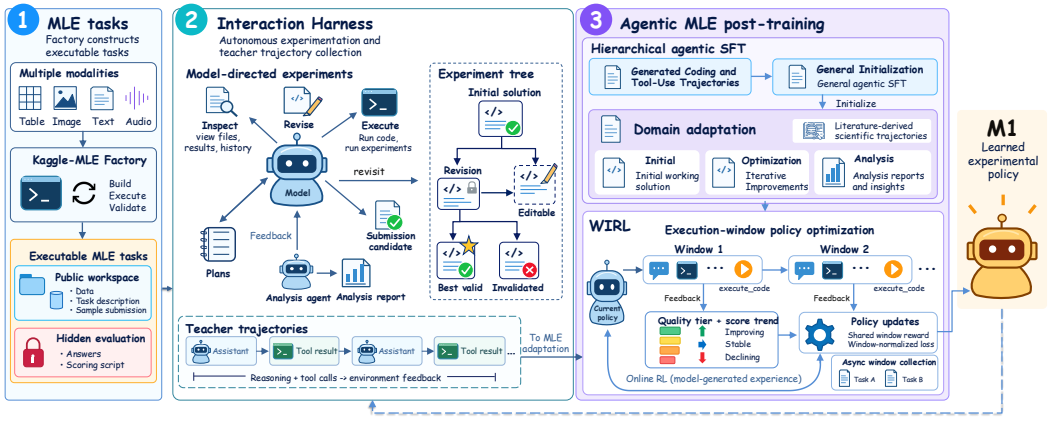


Figure 2: **Overview of M1 training for autonomous MLE.** The Kaggle-MLE Factory constructs executable tasks, and the interaction harness supports model-directed experimentation and trajectory collection. Hierarchical agentic SFT learns from demonstrations, while WIRL further optimizes the policy from execution feedback collected during online interaction.

$y_e^\star \in \arg\max_{y \in \mathcal{Y}_e} S_e(y)$, where $S_e(y)$ denotes the task-specific score under metric m_e, normalized so that higher values indicate better performance. Following MLE-bench (Chan et al., 2025), the agent has access to the task description and public data, and improves solutions through code execution and self-constructed validation. The final submission is independently evaluated using hidden test labels.

In Agentic MLE, an LLM policy π_θ autonomously organizes the optimization process. Given a task e, the model constructs, executes, and improves candidate solutions through continuous tool interaction within the time budget B_e. A trajectory of length L is represented as $\tau = (o_0, a_0, o_1, a_1, \ldots, a_{L-1}, o_L)$, where o_0 is the initial task observation, a_l is the model action at step l, and o_{l+1} is the observation returned after executing a_l. Each action may include model reasoning and one or more tool calls. At step l, the model samples $a_l \sim \pi_\theta(\cdot \mid c_l)$, where c_l consists of the task information and retained interaction history.

Experimental states, including code, execution records, and candidate solutions, are stored in an external workspace and exposed to the model through tool observations. Rather than following a predefined workflow, the model decides when to analyze, revise, execute, or change the optimization direction based on the current state. The model maintains a submission candidate throughout the interaction and selects its final solution $\hat{y}_\tau$ at the end of trajectory τ based on the experimental results and analysis available during interaction. The objective of Agentic MLE is to maximize the expected quality of this selected solution

$$J(\pi_\theta; e) = \mathbb{E}_{\tau \sim p_{\pi_\theta}(\cdot \mid e, B_e)} \left[S_e(\hat{y}_\tau) \right], \tag{1}$$

where p_{π_θ} denotes the trajectory distribution induced by the policy and the interaction environment.

3 Task Construction and Long-Horizon Interaction

As shown in Figure 2, we build the **Kaggle-MLE Factory** to provide executable tasks and the **Interaction Harness** to support model-directed long-horizon experimentation and state management, together enabling hierarchical agentic SFT and WIRL for post-training M1 in autonomous MLE.

3.1 Kaggle-MLE Factory: Constructing Executable MLE Tasks

Long-horizon Agentic MLE training requires executable tasks covering diverse data modalities, objectives, and domains. Historical Kaggle competitions provide a rich source of real-world MLE problems, but their heterogeneous data organization, submission formats, and evaluation protocols make manual task adaptation difficult to scale. Therefore, we build the Kaggle-MLE Factory, which converts historical competitions into executable tasks with a public workspace, a hidden test split, and a unified evaluation interface. The Factory uses a multi-agent pipeline for task construction, validation, and repair. The discovery agent filters competitions by status, scale, and submission

format. After competition data, leaderboard information, and official descriptions are collected, the analyzer agent extracts data structures, fields, and modality characteristics to form a task specification. The generator agent then prepares task data, constructs a held-out test split with hidden labels from the original labeled data, and implements the task-specific evaluator. The task is executed on real data and iteratively repaired based on validator checks of data integrity, submission format, and evaluation behavior to produce valid executable tasks. Standardized task packages support different MLE agent frameworks. Final manual review excludes tasks with unresolved label or scoring issues before trajectory collection. Appendix A details agent responsibilities and implementation.

3.2 Interaction Harness for Long-Horizon MLE Optimization

To support model-directed long-horizon MLE optimization, we design an interaction harness that follows ReAct-style loop (Yao et al., 2022), leaving branch selection and optimization-path decisions to the model rather than an external search policy. The harness stores experimental states in solution nodes, each containing candidate code, execution logs, and metadata. Parent-linked revisions form an experiment tree that the model can revisit and compare to decide whether to continue, backtrack, or change optimization direction. The harness provides tools for solution analysis, code revision, experiment execution, history retrieval, and submission management. The model launches candidate execution through `execute_code` and updates its submission candidate through `update_answer`. The complete tool list and descriptions are provided in Appendix B.

As interaction history approaches the context limit, earlier messages are summarized while recent interactions are retained. For complex analysis, `analyze` delegates task analysis, data exploration, and result diagnosis to an analysis sub-agent with independent context, returning only a summary to the main model. Since overfitting or data leakage can inflate validation scores, the model can call `invalidate_node` based on experimental evidence and analysis to exclude unreliable candidates. With new evidence, the model can update its submission candidate while exploring alternatives.

4 Agentic MLE Post-Training

4.1 Hierarchical Agentic SFT

Long-horizon MLE optimization requires general tool-use, coding skills, and domain-specific decision making. We therefore use two-stage hierarchical agentic SFT. In the first stage, coding and tool-use trajectories from diverse agentic tasks are used to learn basic tool-use and code revision capabilities. The second stage adopts teacher trajectories from MLE domain and literature-derived scientific tasks to improve algorithm design, experimental analysis, and iterative optimization.

Complementary MLE demonstrations. MLE adaptation data cover both task initialization and sustained optimization. For trajectories whose first complete solution executes successfully and reaches a quality threshold, we extract the interaction prefix from task start through the corresponding execution feedback as an *initial-solution trajectory*, typically covering task understanding, algorithm design, and code generation. We also use *long-horizon optimization trajectories* from complete interactions, covering solution construction, feedback analysis, iterative revision, and final submission selection. In addition, analysis sub-agent trajectories are adopted to train the model for independent data analysis and result diagnosis.

Outcome-guided supervision selection. Long-horizon MLE trajectories can contain effective exploration and failure recovery, but may also include misguided decisions or continued degradation after reaching a strong solution. Therefore, we use execution results of submitted solutions to select high-quality trajectories and the segments used for supervision. For trajectories that pass a quality threshold, we identify the first best-performing submission and measure the subsequent performance degradation. When the post-peak degradation exceeds a threshold, we use the first best-performing submission as the anchor for retaining a trajectory prefix; otherwise, we use the last submission. The prefix ends at a complete tool-interaction boundary near the anchor. This removes degraded tails while preserving useful exploration and failure recovery before the peak.

Supervision samples and training objective. To preserve multi-turn context dependencies, each supervision sample combines existing interaction context with a continuous sequence of subsequent model actions and environment feedback from the selected MLE interactions. When context com-

paction occurs, the retained history and summary form the conditioning context for subsequent samples. Training loss is applied only to newly generated reasoning, text responses, and tool-call tokens, while tool feedback and inherited interaction history are used only as conditioning context and masked out from the loss. Details of outcome-guided filtering, score normalization, and the SFT objective appear in Appendix C.1.

4.2 Windowed Interaction Reinforcement Learning

Building on SFT, we further optimize the long-horizon decision policy using execution feedback from the model's own exploration. In MLE, solution quality is available only after training and evaluation, and each result depends on multiple rounds of analysis, code revision, and tool interaction. We introduce Windowed Interaction RL (WIRL), which groups consecutive model decisions made between successive experiment results into an execution window u_m. Each window is conditioned on the accumulated interaction history and associated with the outcome of the corresponding experiment launched through the `execute_code` tool (Sec. 3.2). The first window starts from the initial task state. This allows policy updates to use feedback from completed experiments without waiting for the entire task trajectory to finish.

Window rewards and shared optimization weights. MLE tasks use different evaluation metrics and score scales, so we use task-specific medal thresholds as a common reference for solution quality and compare each valid submission with the most recent valid submission in the trajectory to capture local optimization progress. The reward of execution window u_m is

$$R_m = \begin{cases} -1.0, & \text{execution error caused by model actions,} \\ -0.6, & \text{no such error but no scorable submission,} \\ -0.4 + T_m, & \text{valid submission below the medal threshold,} \\ 1.0 + T_m, & \text{bronze or silver medal,} \\ 2.0 + T_m, & \text{gold medal.} \end{cases} \tag{2}$$

Here, T_m denotes the local performance trend and is set to 0 for the first valid submission. For subsequent valid submissions, $T_m \in \{-0.2, 0, 0.2\}$ according to whether performance decreases, remains unchanged, or improves relative to the previous valid submission, accounting for the direction of the task metric. Unlike standard GRPO (Shao et al., 2024), WIRL directly uses the verifiable execution reward as the shared optimization weight $\widehat{A}_m = R_m$, without group-relative normalization, and applies it to all main-model output tokens included in training within window u_m.

Asynchronous collection and policy correction. Since experiment durations vary, WIRL asynchronously collects windows with trusted execution feedback to form training batches. A long trajectory may span multiple policy updates, so its model calls may be generated by rollout policies that differ from the current training policy. We therefore apply token-level policy correction using log-probabilities recorded during collection. For output token t from model call j in window u_m, let $\ell_{\theta,mjt}$, ℓ^{old}_{mjt}, and ℓ^{roll}_{mjt} denote its log-probabilities under the current training policy, the policy fixed before the update, and the rollout policy, respectively:

$$q_{mjt} = \exp\big(\ell^{\text{old}}_{mjt} - \ell^{\text{roll}}_{mjt}\big)\,, \qquad c_{mjt} = \text{sg}(q_{mjt}\mathbf{1}[q_{\min} \le q_{mjt} \le q_{\max}])\,, \tag{3}$$

where sg denotes stop-gradient, $\mathbf{1}[\cdot]$ is the indicator function, and $q_{\min}$ and $q_{\max}$ are the lower and upper correction bounds. Tokens within $[q_{\min}, q_{\max}]$ are weighted by q_{mjt}, while those outside the interval are excluded from the policy-gradient update.

Policy Optimization Objective. Let L^{PG}_{mjt} denote the PPO-style token-level policy-gradient loss (Schulman et al., 2017) with shared window weight $\widehat{A}_m = R_m$. Because execution windows vary in the number of model calls and output tokens, we normalize the loss by the number of valid output tokens within each window before averaging across windows. Let M_{mjt} denote the training mask for newly generated main-model output tokens and $N_m = \sum_{j,t} M_{mjt}$ the number of valid output tokens in window u_m. For a batch $\mathcal{B}$ containing B windows, we minimize

$$\mathcal{L}_{\mathcal{B}}(\theta) = \frac{1}{B} \sum_{m \in \mathcal{B}} \frac{1}{\max(N_m, 1)} \sum_{j,t} M_{mjt} \left[c_{mjt} L^{\text{PG}}_{mjt} + \lambda_{\text{KL}} K_{mjt} - \lambda_H H_{mjt} \right], \tag{4}$$

Table 1: **Main results on MLE-Bench Lite.** We report performance (%) under the unified M1 Harness and, where available, under general-purpose agentic coding harnesses. Bold and underline mark the best and second-best distinct values in each metric column, including ties. Detailed evaluation results are provided in Appendix D.1.

Model	M1 Harness					General-Purpose Harness	
	Med+	Gold	Silver	Bronze	Medal	Harness	Medal
Same-scale baselines (35B total, ∼3B active)							
Qwen3.6-35B-A3B	62.1	22.7	7.6	<u>7.6</u>	37.9	Claude Code	42.4
Ornith-1.5-35B-A3B	57.6	22.7	12.1	4.5	39.4	Claude Code	47.0
Nex-N2.5-mini	66.7	24.2	13.6	**9.1**	47.0	Claude Code	36.4
Larger frontier models							
DeepSeek-V4-Pro Preview	86.4	40.9	13.6	**9.1**	63.6	–	–
MiniMax M3	81.8	54.5	13.6	0.0	68.2	Codex	54.5
GLM-5.3	<u>90.9</u>	<u>59.1</u>	13.6	0.0	<u>72.7</u>	Claude Code	**77.3**
Kimi-K3	**95.5**	**63.6**	9.1	0.0	<u>72.7</u>	Claude Code	<u>72.7</u>
GPT-5.5 xhigh	81.8	45.5	**22.7**	4.5	<u>72.7</u>	Codex	68.2
GPT-5.6 Sol	<u>90.9</u>	<u>59.1</u>	13.6	4.5	**77.3**	Codex	<u>72.7</u>
M1 (Ours)	84.8	50.0	<u>16.7</u>	6.1	<u>72.7</u>	Claude Code	69.7

where K_{mjt} is the KL regularization term relative to a fixed SFT reference model, H_{mjt} is the entropy of the current policy, and λ_{KL} and λ_H are their coefficients. The full policy loss definition and implementation details are provided in Appendix C.2.

5 Experiments

5.1 Experimental Setup

Benchmarks. We primarily evaluate our trained models on MLE-Bench Lite (Chan et al., 2025), a 22-task subset of MLE-Bench consisting of historical Kaggle competitions for end-to-end machine learning engineering. We report the percentage of tasks awarded each medal tier (Gold, Silver, and Bronze), the percentage receiving any medal (Any Medal), and the percentage exceeding the median human score (Med+), with Any Medal as the primary metric. All five metrics are computed over the 22 tasks and reported as percentages. We also evaluate on NatureBench (Wang et al., 2026), following the ten-task Lite subset used in Frontis-MA1 (Yang et al., 2026a).

Baselines. We compare M1 against two groups of models: (1) *same-scale models*, including Qwen3.6-35B-A3B (Qwen Team, 2026a), Ornith-1.5-35B-A3B (Ornith Team, 2026), and Nex-N2.5-mini (Nex-AGI, 2026); and (2) *larger frontier models*, including DeepSeek-V4-Pro Preview (DeepSeek-AI et al., 2026), MiniMax M3 (MiniMax, 2026), GLM-5.3 (GLM-5-Team et al., 2026), Kimi-K3 (Kimi Team et al., 2026), GPT-5.5 (OpenAI, 2026a), and GPT-5.6 Sol (OpenAI, 2026b).

Implementation details. Each MLE task is assigned a maximum of 3,000 interaction steps and a 12-hour runtime, using 20 vCPUs, 220 GB of RAM, and a single NVIDIA H200 GPU. M1 is initialized from Qwen3.6-35B-A3B. General SFT uses coding and tool-use trajectories from StepFun (Huang et al., 2026). For MLE SFT, we collect teacher trajectories from Kaggle-MLE Factory tasks, MLE-Dojo (Qiang et al., 2025) tasks, and scientific tasks derived from Nature-family papers. Any training tasks overlapping with MLE-Bench or NatureBench (Wang et al., 2026) are excluded. Data statistics, training hyperparameters, and implementation details are provided in Appendix C.

5.2 Main Results

M1 achieves the highest medal rate among same-scale models and reaches the performance level of larger frontier models. As shown in Table 1, under the unified M1 Harness, M1 raises the medal rate of its Qwen3.6-35B-A3B base model from 37.9% to 72.7%, achieving the highest rate

Table 2: **Results on NatureBench Lite.** M denotes Match-SOTA ($g \geq 0$), while S denotes Surpass-SOTA ($g > 0.1$), which serves as a key metric for directly assessing agents' ability to surpass published scientific results.

Model	Harness	M (%) ↑	S (%) ↑
Same-scale baselines			
Qwen3.6-35B-A3B	Claude Code	40.0 (4/10)	0.0 (0/10)
Nex-N2.5-mini	Claude Code	80.0 (8/10)	20.0 (2/10)
Ornith-1.5-35B-A3B	Claude Code	60.0 (6/10)	30.0 (3/10)
Larger frontier models			
DeepSeek-V4-Pro Preview	Claude Code	60.0 (6/10)	20.0 (2/10)
MiniMax M3	Claude Code	70.0 (7/10)	30.0 (3/10)
GPT-5.5	Codex	**100.0** (10/10)	40.0 (4/10)
GLM-5.3	Claude Code	90.0 (9/10)	50.0 (5/10)
Kimi-K3	Claude Code	90.0 (9/10)	**70.0** (7/10)
M1 (Ours)	Claude Code	70.0 (7/10)	50.0 (5/10)

among the same-scale models evaluated and performance comparable to GLM-5.3 and Kimi-K3. These results show that long-horizon agentic post-training substantially improves the base model's MLE capability, bringing a 35B model to a performance level comparable to larger frontier models.

M1 retains its advantage over the base model under a general-purpose coding harness. With Claude Code, M1 achieves a 69.7% medal rate, compared with 42.4% for its base model, and exceeds the strongest evaluated same-scale baseline by 22.7 percentage points. The consistent advantage over the base model across both harnesses suggests that the improvement from post-training is not specific to the M1-Harness and remains effective in a general-purpose agentic coding environment.

On NatureBench Lite, M1 improves scientific task performance over the base model. Table 2 shows that, with Claude Code, M1 raises the Surpass-SOTA rate from 0.0% to 50.0%, exceeding Nex-N2.5-mini at 20.0% and matching GLM-5.3 at 50.0%. Its Match-SOTA rate also improves from 40.0% to 70.0%. These results show that post-training improves scientific task optimization, enabling M1 to match or exceed published reference results on more tasks.

5.3 Ablation Studies

Table 3: Effect of post-training stages on MLE-Bench Lite with the M1 Harness.

Training stage	Any Medal (%) ↑
Base	37.9
+ General Agent SFT	47.0
+ MLE SFT	62.1
+ WIRL	**72.7**

Table 4: RL comparisons on MLE-Bench Lite. Methods are initialized from the same MLE-SFT model.

Method	Any Medal (%) ↑
GRPO	56.1
PPO-style	60.6
WIRL w/o trend reward	66.7
WIRL w/o policy correction	65.2
WIRL	**72.7**

Effect of post-training stages. Table 3 compares successive training stages under the M1 Harness. General Agent SFT raises the average medal rate from 37.9% to 47.0%, MLE-domain SFT further increases it to 62.1%, and WIRL reaches 72.7%. Each stage provides an additional gain. In particular, hierarchical SFT substantially improves MLE performance over the base model, while WIRL further improves performance beyond teacher-supervised training through online execution feedback.

RL baselines and key WIRL design choices. We compare WIRL with trajectory-level GRPO and PPO-style optimization under the same MLE environment and MLE-SFT initialization. GRPO uses group-relative advantages, whereas PPO-style optimization uses a clipped objective weighted by terminal trajectory rewards without an additional critic. We also evaluate WIRL variants that remove the trend reward or policy correction separately. As shown in Table 4, GRPO and PPO-style optimization achieve 56.1% and 60.6%, respectively, both below the 62.1% achieved by MLE-SFT, supporting WIRL's overall design of learning from execution-window feedback. In contrast, the two

WIRL variants reach 66.7% and 65.2%, respectively, while full WIRL further improves the medal rate to 72.7%, indicating that both the local trend reward and policy correction contribute to the final improvement. Further analysis in Appendix D.1 shows that asynchronous execution-window collection reduces the learner waiting fraction compared with trajectory-based GRPO.

5.4 Case Study: Long-Horizon Optimization

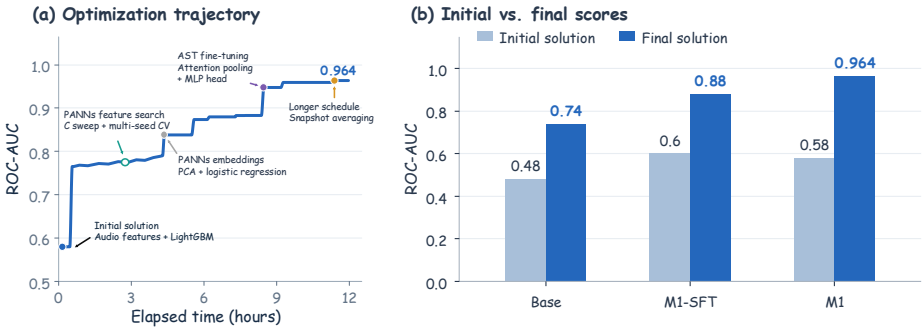


Figure 3: Long-horizon optimization on MLSP 2013 Birds. (a) Representative optimization progress of M1. (b) Initial and final ROC-AUC for Base, M1-SFT, and M1.

To examine the long-horizon optimization behavior of M1, we select MLSP 2013 Birds as a representative case. Figure 3(a) traces the optimization process from audio features and LightGBM through PANNs representations to AST fine-tuning with attention pooling and an MLP head, followed by training-schedule adjustments and snapshot averaging. As shown in Figure 3(b), M1 and M1-SFT start with solutions of similar quality, but M1 improves ROC-AUC from 0.58 to 0.964, compared with 0.60 to 0.88 for M1-SFT. This case shows how M1 continues to improve solutions through sustained experimentation and execution feedback.

5.5 Generalization to Other Optimization Tasks

Agent scaffold optimization. Following ShinkaEvolve (Lange et al., 2026), we optimize a Python scaffold for a fixed GPT-4.1-nano solver on AIME 2025 (Zhang & Math-AI, 2025) and evaluate it on AIME problems from years not used for optimization. Table 5 shows that the scaffold optimized by M1 achieves 43.7% held-out accuracy, compared with 23.3% for the scaffold optimized by the base model. These results show that post-training improves the ability to design effective agent workflows, with optimization gains extending to unseen problems. Full results and evaluation protocols are provided in Appendix D.3.

Table 5: AIME scaffold optimization with GPT-4.1-nano. Higher accuracy (%) and fewer calls are better. Bold/underline: best/second-best.

Method	Dev. ↑	Held-out ↑	Calls ↓
Initial	21.1	20.0	**1.00**
Majority@5	24.4	24.8	5.00
Base	28.9	23.3	5.74
Kimi-K3	45.6	40.0	8.65
GLM-5.3	**47.8**	**48.1**	9.39
M1-SFT (Ours)	37.8	38.1	<u>3.51</u>
M1 (Ours)	<u>46.7</u>	<u>43.7</u>	8.40

Algorithm-component optimization on MLS-Bench-Lite. On MLS-Bench-Lite (Lyu et al., 2026), we optimize algorithmic components within fixed research pipelines on 12 tasks, one from each domain, under the same five-hour budget. Figure 4 shows that M1 improves the mean verifier reward from 0.321 to 0.403, outperforming Base on ten tasks, tying on one, and scoring lower on one. The gains span language-model demasking, multi-objective optimization, and causal discovery, demonstrating transfer beyond end-to-end MLE to component optimization across scientific domains. Full protocols and task-level results are provided in Appendix D.4.

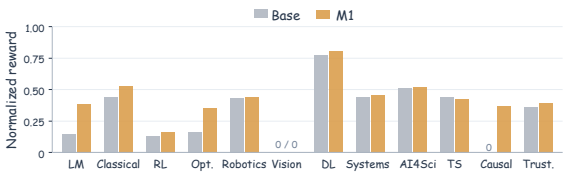


Figure 4: Algorithm-component optimization across 12 MLS-Bench-Lite domains. Bars show normalized verifier rewards for Base and M1.

6 Related work

6.1 Agents for Automated ML Research

LLM agents increasingly automate data analysis, model development, and experimental iteration, with recent work extending to autonomous AI system design and improvement (Wu et al., 2026). AIDE (Jiang et al., 2025) and MLE-STAR (Nam et al., 2025) improve candidate solutions through iterative search and refinement, while MLEvolve (Du et al., 2026) and MARS (Chen et al., 2026) use structured tree- or graph-based search, memory, and planning to organize long-horizon experimentation. R&D-Agent (Yang et al., 2025) studies multi-role collaboration, while InternAgent (Feng et al., 2026) and FM-Agent (Li et al., 2025) extend autonomous AI research across machine learning, algorithm optimization, and engineering tasks. Recent systems including Arbor (Jin et al., 2026) and Argus (Li et al., 2026a) maintain persistent research states and increase model control.

Recent work develops MLE capabilities by training models on interaction experience. ML-Agent (Liu et al., 2025) combines multi-turn demonstrations with single-step RL, while RL for MLE Agents (Yang et al., 2026b), MLE-RL (Li et al., 2026b), and AceGRPO (Cai et al., 2026) study online MLE training with execution feedback, asynchronous sampling, or intermediate trajectory states. Learning to Ideate (Zhang et al., 2026) trains models to generate algorithm improvement suggestions, whereas Frontis-MA1 (Yang et al., 2026a) applies SFT and RL to program evolution operators and combines them through an external evolutionary framework. These approaches primarily train individual interaction steps or local components of externally organized optimization workflows. SandMLE (Zhou et al., 2026) applies trajectory-wise on-policy RL in micro-scale synthetic MLE environments. We train model-directed long-horizon interaction policies on real-world MLE tasks, with WIRL using execution windows to learn multi-turn decisions from delayed execution feedback under asynchronous updates.

6.2 Post-Training Long-Horizon Agentic Models

Recent research has increasingly focused on post-training agentic models for sustained interaction, complex tool use, and long-horizon decision making. SWE-Gym (Pan et al., 2024) trains coding agents using trajectories collected from executable environments, while Search-R1 (Jin et al., 2025) and WebRL (Qi et al., 2025) learn multi-turn tool use from environment feedback and outcome rewards. AgentFlow (Li et al., 2026c) directly optimizes planning policies in general multi-turn agentic interaction and uses trajectory-level outcomes to train intermediate decisions. Recent work has also explored generalist agentic models for long-horizon interaction. Agents-A1 (Bai et al., 2026) uses long-horizon knowledge-action trajectories and multi-stage training, together with multi-teacher on-policy distillation (OPD), to combine capabilities from multiple domain teachers into a unified model. Ornith-1.5 (Ornith Team, 2026) extends self-scaffolding by jointly optimizing task generation, scaffold construction, and solution rollouts through RL.

7 Conclusion

In this work, we introduce M1, a 35B MoE agentic model trained to autonomously conduct and improve machine learning experiments over long interaction horizons. We build an automated MLE task factory and a model-directed interaction harness, and train M1 with hierarchical agentic SFT and WIRL to learn from teacher demonstrations and execution feedback collected from its own exploration. Extensive experiments show that M1 improves over its base model on MLE-Bench Lite and achieves performance comparable to larger frontier models. Further improvements on NatureBench Lite and in agent scaffold and algorithm optimization demonstrate cross-task generalization. These results support treating long-horizon optimization decision making as a trainable model capability, providing a practical path toward autonomous machine learning engineering and scientific experimentation.

AI USE STATEMENT

We used large language models to assist with manuscript translation and language editing, as well as experimental code implementation. The authors reviewed and revised the assisted text and code and take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Task construction is described in Section 3.1 and Appendix A. Training methods and key hyperparameters are detailed in Sections 4.1 and 4.2 and Appendix C. Evaluation protocols and detailed benchmark results are provided in Section 5.1 and Appendix D. We will publicly release the M1 model weights upon acceptance of the paper.

REFERENCES

Lei Bai, Zongsheng Cao, Yang Chen, Zhiyao Cui, Shangheng Du, Yue Fan, Shiyang Feng, Zijie Guo, Haonan He, Liang He, et al. Scaling the horizon, not the parameters: Reaching trillion-parameter performance with a 35b agent. *arXiv preprint arXiv:2606.30616*, 2026.

Yuzhu Cai, Zexi Liu, Xinyu Zhu, Cheng Wang, Yanfeng Wang, and Siheng Chen. Acegrpo: Adaptive curriculum enhanced group relative policy optimization for autonomous machine learning engineering. *arXiv preprint arXiv:2602.07906*, 2026.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. Mlebench: Evaluating machine learning agents on machine learning engineering. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

Jiefeng Chen, Bhavana Dalvi Mishra, Jaehyun Nam, Rui Meng, Tomas Pfister, and Jinsung Yoon. Mars: Modular agent with reflective search for automated ai research. *arXiv preprint arXiv:2602.02660*, 2026.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Shangheng Du, Xiangchao Yan, Jinxin Shi, Zongsheng Cao, Shiyang Feng, Zichen Liang, Boyuan Sun, Tianshuo Peng, Yifan Zhou, Xin Li, et al. Mlevolve: A self-evolving framework for automated machine learning algorithm discovery. *arXiv preprint arXiv:2606.06473*, 2026.

Shiyang Feng, Runmin Ma, Xiangchao Yan, Yue Fan, Yusong Hu, Songtao Huang, Shuaiyu Zhang, Zongsheng Cao, Tianshuo Peng, Jiakang Yuan, et al. Internagent-1.5: A unified agentic framework for long-horizon autonomous scientific discovery. *arXiv preprint arXiv:2602.08990*, 2026.

GLM-5-Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Ailin Huang, Ang Li, Aobo Kong, Bin Wang, Binxing Jiao, Bo Dong, Bojun Wang, Boyu Chen, Brian Li, Buyun Ma, et al. Step 3.5 flash: Open frontier-level intelligence with 11b active parameters. *arXiv preprint arXiv:2602.10604*, 2026.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Jiajie Jin, Yuyang Hu, Kai Qiu, Qi Dai, Chong Luo, Guanting Dong, Xiaoxi Li, Tong Zhao, Xiaolong Ma, Gongrui Zhang, et al. Toward generalist autonomous research via hypothesis-tree refinement. *arXiv preprint arXiv:2606.11926*, 2026.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Robert Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations*, volume 2026, pp. 74026–74078, 2026.

Annan Li, Chufan Wu, Zengle Ge, Yee Hin Chong, Zhinan Hou, Lizhe Cao, Cheng Ju, Jianmin Wu, Huaiming Li, Haobo Zhang, et al. The fm agent. *arXiv preprint arXiv:2510.26144*, 2025.

Boxiu Li, Zimo Wen, Yijia Fan, Junxiang Lei, Sufeng Guo, Jiaao Wu, Ruize Tang, Mukai Li, Yifei Shen, Xiaoyu Chen, et al. Argus: A general-purpose agentic runtime for long-horizon reasoning. *arXiv preprint arXiv:2608.05144*, 2026a.

Yujiang Li, Zhenyu Hou, Xiaohan Jia, Zihan Luo, Zhilei Bei, Rui Lu, Hong Huang, Jie Tang, and Yuxiao Dong. MLE-RL: Reinforcement learning for self-improvement in machine learning agents, 2026b. URL https://openreview.net/forum?id=nElqyHPHAz.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Y Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use. In *International Conference on Learning Representations*, volume 2026, pp. 50524–50570, 2026c.

Zexi Liu, Jingyi Chai, Xinyu Zhu, Shuo Tang, Rui Ye, Bo Zhang, Lei Bai, and Siheng Chen. Ml-agent: Reinforcing llm agents for autonomous machine learning engineering. *arXiv preprint arXiv:2505.23723*, 2025.

Bohan Lyu, Yucheng Yang, Siqiao Huang, Jiaru Zhang, Qixin Xu, Xinghan Li, Xinyang Han, Yicheng Zhang, Huaqing Zhang, Runhan Huang, et al. Mls-bench: A holistic and rigorous assessment of ai systems on building better ai. *arXiv preprint arXiv:2605.08678*, 2026.

MiniMax. MiniMax M3: Frontier coding, 1M context, native multimodality—all in one model, June 2026. URL https://www.minimax.io/blog/minimax-m3.

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. *arXiv preprint arXiv:2506.15692*, 2025.

Nex-AGI. Nex-N2.5-mini. Hugging Face model card, 2026. URL https://huggingface.co/nex-agi/Nex-N2.5-mini.

OpenAI. Introducing GPT-5.5, April 2026a. URL https://openai.com/index/introducing-gpt-5-5/.

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

Ornith Team. Ornith-1.5: From self-scaffolding to self-improvement, 2026. URL https://ornith.ai/ornith_1_5.html.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 79791–79821, 2025.

Rushi Qiang, Yuchen Zhuang, Yinghao Li, Dingu Sagar VK, Rongzhi Zhang, Changhao Li, Ian Wong, Sherry Yang, Percy Liang, Chao Zhang, et al. Mle-dojo: Interactive environments for empowering llm agents in machine learning engineering. *Advances in Neural Information Processing Systems*, 38, 2025.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026a. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL https://qwen.ai/blog?id=qwen3.8.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Yuru Wang, Lejun Cheng, Yuxin Zuo, Sihang Zeng, Bingxiang He, Che Jiang, Junlin Yang, Yuchong Wang, Kaikai Zhao, Weifeng Huang, et al. Naturebench: Can coding agents match the published sota of nature-family papers? *arXiv preprint arXiv:2606.24530*, 2026.

Kai Wu, Hao Lyu, Zhen Luo, Chaofan Wang, Siyu Ye, Jinghao Lin, Xiaozhong Ji, Boyuan Jiang, Shengzhi Wang, Zihan Wang, et al. Ai4ai survey: From long-horizon agents to recursive self-improvement—definitions, reliable horizons, and open problems. *Preprints*, 2026.

Junlin Yang, Che Jiang, Yu Fu, Tianwei Luo, Can Ren, Weizhi Wang, Kaikai Zhao, Hongyi Liu, Yuxin Zuo, Yuru Wang, et al. Frontis-ma1: Training an ai4ai model towards recursive self-improvement in machine learning engineering. *arXiv preprint arXiv:2607.28568*, 2026a.

Sherry Yang, Joy He-Yueya, and Percy Liang. Reinforcement learning for machine learning engineering agents. In *International Conference on Learning Representations*, volume 2026, pp. 156989–157011, 2026b.

Xu Yang, Xiao Yang, Shikai Fang, Bowen Xian, Yuante Li, Jian Wang, Minrui Xu, Haoran Pan, Xinpeng Hong, Weiqing Liu, et al. R&d-agent: Automating data-driven ai solution building through llm-powered automated research, development, and evolution. *arXiv preprint arXiv:2505.14738*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Z.ai. GLM-5.2: Built for long-horizon tasks, June 2026. URL https://z.ai/blog/glm-5.2.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (AIME) 2025, 2025. URL https://github.com/math-ai-org/aime25.

Yunxiang Zhang, Kang Zhou, Zhichao Xu, Kiran Ramnath, Yun Zhou, Sangmin Woo, Haibo Ding, and Lin Lee Cheong. Learning to ideate for machine learning engineering agents. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 436–447, 2026.

Yuhang Zhou, Lizhu Zhang, Yifan Wu, Jiayi Liu, Xiangjun Fan, Zhuokai Zhao, and Hong Yan. Synthetic sandbox for training machine learning engineering agents. *arXiv preprint arXiv:2604.04872*, 2026.

A Details of the Kaggle-MLE Factory

We begin with a Meta-Kaggle[1] competition metadata snapshot containing 11,649 entries. Guided by the task-selection principles of MLE-bench (Chan et al., 2025), the discovery agent considers task relevance, data availability, and the feasibility of offline evaluation. Automated initial screening uses competition completion, at least 50 participating teams, and a dataset size limit of 200 GB as default criteria, checks submission-interface compatibility, and consults an exclusion list of existing tasks. The agent further selects candidates across domains and data modalities, including vision, text, audio, tabular data, and time series. Candidates are collected and processed in multiple batches.

The downloader and scraper retrieve competition data, official descriptions, and leaderboard information. The analyzer agent extracts data structures, field information, and modality characteristics to form a task specification, which the generator agent uses to produce data-preparation and scoring code. When test labels are unavailable, the pipeline constructs a held-out split from the original labeled data and keeps its labels on the evaluation side. The executor runs generated task packages on real data, while the validator agent checks structural completeness, submission-format consistency, non-degenerate labels, train–test identifier overlap, and data availability. The validator also executes the scoring function to check that it produces a valid score. Execution and validation errors are returned to the construction agents for repair. We conduct a final human review of the 160 registered tasks, excluding six with unresolved label or scoring-implementation issues and retaining 154 tasks for subsequent trajectory collection.

B Interaction Harness Tools

The Interaction Harness described in Section 3.2 provides tools for inspecting the task environment, revising candidate solutions, running experiments, retrieving previous states, and managing submissions. The model selects subsequent actions based on execution feedback; Table 6 summarizes the available tools and their purposes.

Table 6: Tools provided by the Interaction Harness.

Tools	Purpose
`analyze`	Delegate data, result, or error analysis
`execute_bash`	Inspect data and runtime environment
`write_full_code`, `patch_code`	Create or revise candidate solutions
`execute_code`, `control_run`	Launch and control experiments
`write_plan`, `edit_plan`, `read_plan`	Maintain persistent experiment plans
`list_nodes`, `select_node`	Retrieve candidate states and history
`invalidate_node`	Exclude candidates with untrustworthy metrics
`update_answer`, `get_current_answer`	Maintain valid submission candidates

C Agentic MLE Post-Training Details

C.1 Details of Hierarchical Agentic SFT

Training data sources. The first SFT stage uses general coding and tool-use trajectories from StepFun (Huang et al., 2026), while the second stage combines MLE teacher trajectories with literature-derived scientific trajectories. The MLE tasks comprise 154 tasks constructed by the Kaggle-MLE Factory and 134 tasks selected from MLE-Dojo (Qiang et al., 2025), with competitions overlapping with MLE-Bench removed. Following the task-construction approach of NatureBench (Wang et al., 2026), we construct scientific experimentation tasks from 164 papers published in Nature-family journals, covering biology, materials science, physics, chemistry, and medicine, among other fields. The data used in these scientific tasks do not overlap with the NatureBench evaluation set. MLE teacher trajectories are generated by GLM-5.2 (Z.ai, 2026), while scientific experimentation trajectories are generated by GLM-5.2 and Qwen3.8-Max (Qwen Team, 2026b).

Retained corpus and sampling. The corpus supplied to MLE-domain SFT contains 636 retained teacher trajectories: 412 MLE trajectories covering 236 tasks and 224 scientific experimentation

[1] `https://www.kaggle.com/datasets/kaggle/meta-kaggle`

trajectories covering 77 tasks. Of the scientific trajectories, 143 are generated by GLM-5.2 and 81 by Qwen3.8-Max. We count trajectories by distinct source runs before segmentation and repeated sampling; main-agent segments, initial-solution prefixes, and analysis sub-agent interactions from the same run share a single source-trajectory count. The task counts describe coverage of the retained corpus within the task pools above. Table 7 summarizes the source trajectories and training input records.

Table 7: Composition of the MLE-domain SFT input corpus. Source trajectories count distinct retained runs. Input records count serialized training examples after segmentation and repeated sampling, before training-time preprocessing and packing.

Domain	Teacher	Source trajectories	Input records
MLE	GLM-5.2	412	4,961
Scientific experimentation	GLM-5.2	143	342
Scientific experimentation	Qwen3.8-Max	81	236
Total		636	5,539

The MLE input files contain 1,559 main-agent records and 1,843 analysis sub-agent records. The main-agent file is included twice and the sub-agent file once, contributing 4,961 records to the input mixture. The scientific data contain 171 GLM-5.2 records and 118 Qwen3.8-Max records before repetition; each is included twice, contributing 342 and 236 records, respectively. The Qwen3.8-Max data combine two collection batches containing 49 and 32 source trajectories, which yield 66 and 52 records before repetition. These sampling multiplicities determine the input mixture, while the trajectory counts in Table 7 count each source run once.

After training-time preprocessing and sequence packing, the training log records 1,173 sequences with a mean length of approximately 257,843 tokens, corresponding to approximately 302.45 million tokens in the packed dataset. This token count includes conditioning context and generated outputs and describes the dataset before repetition across the eight training epochs. The loss is computed on the output positions specified below.

Outcome-guided supervision selection. For MLE trajectories with a valid leaderboard reference, let $(b_1, \ldots, b_K)$ denote the valid scores obtained in submission order. Here, $b_j \in [0, 1]$ is the submission's relative standing on the task's private leaderboard, accounting for the direction of the evaluation metric so that larger values indicate better performance. We use the same quality threshold $\eta = 0.4$ for trajectory filtering and initial-solution extraction. Trajectories with $\max_j b_j < \eta$ are discarded entirely; those with $\max_j b_j \geq \eta$ pass the quality filter. For retained trajectories, we define the first occurrence of the best score as

$$j^\star = \min \arg\max_j b_j,$$

and measure the average relative degradation after this point as

$$d(\tau) = \begin{cases} \dfrac{b_{j^\star} - \frac{1}{K-j^\star} \sum_{j=j^\star+1}^{K} b_j}{b_{j^\star}}, & j^\star < K, \\ 0, & j^\star = K. \end{cases} \tag{5}$$

Given a degradation threshold $\delta \geq 0$, we use the first best-performing submission as the truncation anchor when $d(\tau) > \delta$; otherwise, we use the last submission. The retained prefix includes the anchor's `update_answer` call and its tool response, followed by a randomly selected extension of up to ten complete tool interactions. The extension stops before the next submission and ends after a tool response. This filtering removes degraded trajectory tails according to their overall post-peak behavior.

The default degradation threshold is $\delta = 0.05$, as reported in Table 8.

Supervision Samples and Training Objective. Let $(w_1, \ldots, w_S)$ denote the supervision samples constructed from a trajectory, and let $x_{s,t}$ be the t-th token in sample w_s. The SFT objective is

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\sum_{s=1}^{S} \sum_{t \in \mathcal{M}(w_s)} \log \pi_\theta \left(x_{s,t} \mid x_{s,<t} \right), \tag{6}$$

where $\mathcal{M}(w_s)$ contains only the positions of newly generated reasoning, text responses, and tool-call tokens in the current segment. Tool feedback and inherited interaction history are used only as conditioning context and are masked out from the loss.

C.2 Details of Windowed Interaction Reinforcement Learning

Local Optimization Objective. Let H_m denote the experimental history at the start of execution window u_m and ζ_m the closed-loop interaction segment within the window. The conditional distribution $P_\theta(\cdot \mid H_m)$ is induced by the policy and the interaction environment, and $R_m(\zeta_m)$ is the reward computed from the corresponding execution result. For a fixed distribution D over window-start histories, we define the local objective as

$$J_{\mathrm{win}}(\theta; D) = \mathbb{E}_{H_m \sim D} \mathbb{E}_{\zeta_m \sim P_\theta(\cdot|H_m)} \left[R_m(\zeta_m) \right]. \tag{7}$$

Local Performance Trend. For the first valid submission, we set $T_m = 0$. For subsequent valid submissions, let s_m and s_{prev} denote the current score and the most recent valid score, respectively. We define the score change according to the metric direction as

$$\Delta_m = \begin{cases} s_m - s_{\mathrm{prev}}, & \text{higher is better,} \\ s_{\mathrm{prev}} - s_m, & \text{lower is better,} \end{cases} \qquad T_m = \begin{cases} 0.2, & \Delta_m > 0, \\ 0, & \Delta_m = 0, \\ -0.2, & \Delta_m < 0. \end{cases} \tag{8}$$

Rollout Log-Probability and Policy Correction Details. For each generated output token, we record the corresponding rollout log-probability $\ell^{\mathrm{roll}}_{mjt}$ during data collection. The correction ratio q_{mjt} in Eq. 3 compares this recorded value with the pre-update training log-probability $\ell^{\mathrm{old}}_{mjt}$. The bounds $q_{\min}$ and $q_{\max}$ define the range in which token-level policy correction is applied. Accordingly, the correction is used at the token level rather than as a trajectory-level importance-sampling ratio.

Token-Level Policy-Gradient Loss. For output token t from model call j in window u_m, we define the probability ratio $r_{mjt} = \exp(\ell_{\theta,mjt} - \ell^{\mathrm{old}}_{mjt})$ between the current and pre-update policies. Using the shared window weight $\widehat{A}_m = R_m$, the PPO-style token-level policy-gradient loss is

$$L^{\mathrm{PG}}_{mjt} = \max\Big(-r_{mjt}\widehat{A}_m,\ -\mathrm{clip}(r_{mjt}, 1-\epsilon_-, 1+\epsilon_+)\widehat{A}_m \Big), \tag{9}$$

where ϵ_- and ϵ_+ control the lower and upper clipping bounds, respectively. This loss is weighted by c_{mjt} in the overall objective in Eq. 4.

Window Normalization and Masking Details. The correction weight c_{mjt} is applied only to the policy-gradient term and does not modify the KL regularization, entropy term, or window-normalization denominator. The training mask M_{mjt} selects newly generated main-model output tokens. For a window with no valid output tokens, the masked token sum is zero, while $\max(N_m, 1)$ prevents division by zero. Padded window slots therefore contribute zero loss, and the outer normalization remains over the B window slots in the batch.

Execution Feedback Filtering. WIRL updates use only windows with trusted execution feedback. Failures caused by infrastructure errors or unavailable scoring signals are excluded from training rather than being assigned the model-error reward in Eq. 2.

C.3 Hyperparameters

Table 8 summarizes the key SFT training settings and the WIRL `medal_absolute_v1` configuration corresponding to the reward formulation in Section 4.2. The supervision thresholds are the defaults of the trajectory-filtering pipeline. Both SFT stages and WIRL use cosine learning-rate decay; WIRL uses no learning-rate warmup.

D Supplementary Experimental Protocols and Results

D.1 MLE-Bench Lite Evaluation Details

Table 9 reports MLE-Bench Lite results with the M1 Harness, including average rates and SEM over three evaluations for M1 and same-scale baselines, and one result for each larger frontier model. All

Table 8: Key hyperparameters for hierarchical agentic SFT and WIRL.

Hyperparameter	Description	Value
General Agent SFT		
Learning rate	Initial / minimum learning rate	2×10^{-5} / 10^{-6}
Batch size	Global batch size	32
Sequence length	Maximum training tokens per sequence	131,072
Training epochs	Number of passes over the dataset	1
Warmup ratio	Fraction of training steps for warmup	0.05
MLE SFT		
Learning rate	Initial / minimum learning rate	10^{-5} / 10^{-6}
Batch size	Global batch size	8
Sequence length	Maximum training tokens per sequence	262,144
Training epochs	Number of passes over the dataset	8
Warmup ratio	Fraction of training steps for warmup	0.05
η	Supervision quality threshold	0.4
δ	Post-peak degradation threshold	0.05
WIRL		
Learning rate	Initial / minimum learning rate	10^{-6} / 10^{-7}
B	Execution windows per training batch	32
Sequence length	Maximum training tokens per sequence	131,072
Sampling	Rollout temperature / top-p / top-k	0.6 / 0.95 / 20
$[q_{\min}, q_{\max}]$	Token-level policy correction interval	$[0.2, 4.0]$
(ϵ_-, ϵ_+)	Lower / upper policy clipping parameters	$(0.3, 3.0)$
λ_{KL}	KL regularization coefficient	10^{-3}
λ_H	Entropy regularization coefficient	10^{-4}

rates are computed over 22 tasks; tasks without a valid scored submission count as no medal. For the general-purpose harness results in Table 1, M1 and the same-scale baselines are each evaluated three times with Claude Code. Any Medal rates (mean $\pm$ SEM) are $42.42\% \pm 4.01\%$ for Qwen3.6-35B-A3B, $46.97\% \pm 3.03\%$ for Ornith-1.5-35B-A3B, $36.36\% \pm 6.94\%$ for Nex-N2.5-mini, and $69.70\% \pm 1.52\%$ for M1. Each larger frontier model has one reported evaluation result. The Kimi-K3 result with Claude Code[2] and the GPT-5.6 Sol result with Codex[3] are from Frontis-MA1 (Yang et al., 2026a); the remaining results are from our evaluations.

Table 9: MLE-Bench Lite results with the M1 Harness. Percentages are reported as average $\pm$ SEM over three evaluations for M1 and same-scale baselines; one result is reported for each larger frontier model.

Model	Med+ (%)	Gold (%)	Silver (%)	Bronze (%)	Any Medal (%)
Qwen3.6-35B-A3B	62.12 ± 4.01	22.73 ± 2.62	7.58 ± 4.01	$\underline{7.58} \pm 1.52$	37.88 ± 5.46
Ornith-1.5-35B-A3B	57.58 ± 1.52	22.73 ± 6.94	12.12 ± 1.52	4.55 ± 2.62	39.39 ± 4.01
Nex-N2.5-mini	66.67 ± 1.52	24.24 ± 6.06	13.64 ± 2.62	$\mathbf{9.09} \pm 4.55$	46.97 ± 1.52
DeepSeek-V4-Pro Preview	86.36	40.91	13.64	**9.09**	63.64
MiniMax M3	81.82	54.55	13.64	0.00	68.18
GLM-5.3	<u>90.91</u>	<u>59.09</u>	13.64	0.00	<u>72.73</u>
Kimi-K3	**95.45**	**63.64**	9.09	0.00	<u>72.73</u>
GPT-5.5 xhigh	81.82	45.45	**22.73**	4.55	<u>72.73</u>
GPT-5.6 Sol	<u>90.91</u>	<u>59.09</u>	13.64	4.55	**77.27**
M1	84.85 ± 3.03	50.00 ± 2.62	$\underline{16.67} \pm 1.52$	6.06 ± 1.52	$\underline{72.73} \pm 2.62$

[2] https://code.claude.com/docs/en/overview

[3] https://learn.chatgpt.com/docs/codex/cli

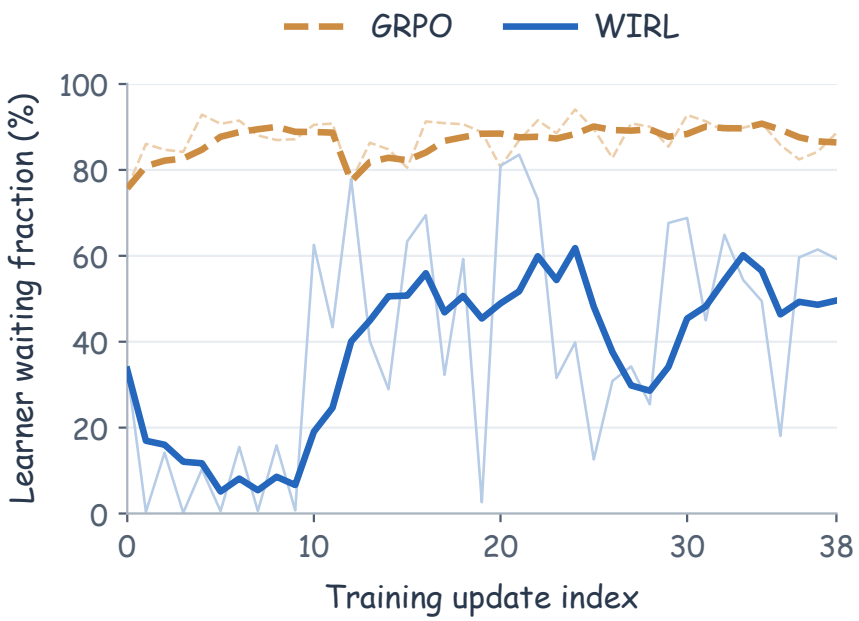


Figure 5: Learner waiting fractions during RL training. Light curves show recorded per-update values; dark curves show trailing five-update means.

Medal composition and harness compatibility. Table 9 shows that post-training improves both medal coverage and the frequency of higher-tier outcomes: the Gold rate rises from 22.73% for the base model to 50.00% for M1, while Any Medal rises from 37.88% to 72.73%. Under Claude Code, M1 retains a substantial advantage over the base model, with 69.7% versus 42.4% Any Medal. Together, these results support improvements in solution quality that remain effective across both task-specific and general-purpose interaction harnesses.

The post-training stage results in Table 3 are averaged over three evaluations. Any Medal rates (mean $\pm$ SEM) are $37.88\% \pm 5.46\%$ for Base, $46.97\% \pm 1.52\%$ for General Agent SFT, $62.12\% \pm 1.52\%$ for MLE SFT, and $72.73\% \pm 2.62\%$ for WIRL. Table 10 expands the RL comparisons in Table 4, reporting average $\pm$ SEM over three evaluations.

Table 10: RL baselines and WIRL ablations on MLE-Bench Lite. Any Medal rates are reported as average $\pm$ SEM over three evaluations.

Method	**Any Medal (%)**
GRPO	56.06 ± 5.46
PPO-style	60.61 ± 4.01
WIRL w/o trend reward	$\underline{66.67} \pm 3.03$
WIRL w/o policy correction	65.15 ± 5.46
WIRL	$\mathbf{72.73} \pm 2.62$

Learner waiting time during RL training. Figure 5 compares the learner waiting fractions of GRPO and WIRL under the same training setup. The waiting fraction is the time spent waiting for training data divided by the sum of waiting and training time. Over the recorded updates, the mean per-update waiting fraction is 87.6% for GRPO and 39.3% for WIRL, a difference of 48.3 percentage points. WIRL exhibits substantially lower learner waiting overhead during RL training.

D.2 NatureBench Lite

We use the ten-task NatureBench Lite subset listed by Frontis-MA1 (Yang et al., 2026a), with the S and M definitions from NatureBench (Wang et al., 2026). We follow the official NatureBench evaluation protocol, with a four-hour runtime budget per task that excludes time spent scoring submissions. Following NatureBench, we compute the normalized relative gap for each evaluation instance by

subtracting the paper-reported SOTA value from the agent's primary metric value, adjusting the sign so that improvements are positive, and dividing by the absolute SOTA value. Instances without a valid submission receive a gap of -1. The task-level score g is the mean gap across instances. Match-SOTA and Surpass-SOTA report the percentages of tasks with $g \geq 0$ and $g > 0.1$, respectively. We report a single evaluation of M1 over the ten tasks, with Surpass-SOTA on five tasks and Match-SOTA on seven tasks, corresponding to 50.0% and 70.0%, respectively. DeepSeek-V4-Pro Preview and MiniMax M3 (MiniMax, 2026) results are taken from Table 2 of Frontis-MA1. Our NatureBench evaluations use NVIDIA H200 GPUs; external reference results retain their original evaluation conditions.

Matching and surpassing scientific reference results. Under Claude Code, M1 improves Surpass-SOTA from 0.0% to 50.0% and Match-SOTA from 40.0% to 70.0% relative to the base model. Compared with Nex-N2.5-mini, M1 has a higher Surpass-SOTA rate, 50.0% versus 20.0%, but a lower Match-SOTA rate, 70.0% versus 80.0%. M1 more frequently exceeds the reference by the Surpass-SOTA margin, while Nex matches the reference on a larger fraction of evaluations. M1 matches the 50.0% Surpass-SOTA rate of GLM-5.3, whose Match-SOTA rate is higher at 90.0%.

D.3 AIME Agent Scaffold Optimization

The outer model optimizes a Python agent scaffold that calls a fixed `gpt-4.1-nano-2025-04-14` solver. All optimizers start from the same single-query scaffold and share the task environment, tools, and evaluator. Each search has a four-hour time limit and a maximum of 1,000 interaction steps, with a context limit of 200,000 tokens and an output limit of 65,536 tokens per step. The scaffold can modify prompting, calling, verification, voting, and answer-extraction procedures, with at most ten inner-model calls per problem. Initial uses one solver call, while Majority@5 takes a majority vote over five independently sampled answers.

Development uses all 30 problems from AIME 2025, with three evaluations per problem for each candidate scaffold. The outer model receives aggregate accuracy, call counts, and sanitized error summaries. Exact-request response caching is used during development to reuse inner-solver responses to identical requests. After search, the selected scaffold is frozen and evaluated using fresh solver calls on AIME 2023, 2024, and 2026. Each held-out year contains 30 problems and is evaluated three times; these years provide no feedback for scaffold development or selection. Thus, each optimizer contributes one frozen scaffold, with three evaluation repetitions per year.

Accuracy is the fraction of answers that exactly match the reference integer answer; missing or incorrectly formatted answers count as incorrect. Table 11 reports development accuracy, accuracy for each held-out year, the mean across the three held-out years, and mean inner-model calls per problem on the held-out evaluations. Each annual accuracy averages the three evaluations of that year's 30 problems.

Table 11: Complete AIME scaffold optimization results. Accuracy (%) is higher-is-better; calls are lower-is-better. Held-out accuracy and calls are averaged over AIME 2023, 2024, and 2026; each year's accuracy averages three evaluations of 30 problems. Bold and underline mark the best and second-best distinct values in each column, including ties.

Method	**Development**	**Held-out accuracy (%)**				**Calls per problem**
	2025	2023	2024	2026	Average	
Initial	21.1	20.0	23.3	16.7	20.0	**1.00**
Majority@5	24.4	18.9	32.2	23.3	24.8	5.00
Base	28.9	18.9	28.9	22.2	23.3	5.74
Ornith-1.5-35B-A3B	24.4	20.0	18.9	16.7	18.5	**1.00**
Nex-N2.5-mini	32.2	17.8	27.8	33.3	26.3	5.93
Kimi-K3	45.6	37.8	41.1	<u>41.1</u>	40.0	8.65
GLM-5.3	**47.8**	**48.9**	**51.1**	**44.4**	**48.1**	9.39
M1-SFT (Ours)	37.8	35.6	42.2	36.7	38.1	<u>3.51</u>
M1 (Ours)	<u>46.7</u>	<u>43.3</u>	<u>43.3</u>	**44.4**	<u>43.7</u>	8.40

Solver-call efficiency after SFT. The scaffold optimized by M1-SFT achieves 38.1% held-out accuracy with 3.51 solver calls per problem, approaching the 40.0% achieved by Kimi-K3 with approximately 59% fewer calls. It also outperforms Majority@5, which reaches 24.8% with five calls per problem. M1-SFT therefore improves held-out accuracy over fixed majority voting while using fewer solver calls, suggesting that its optimized scaffold uses the solver more effectively. These results indicate that SFT already supports effective scaffold optimization across held-out years.

Further gains from WIRL. WIRL raises held-out accuracy from 38.1% to 43.7%, a gain of 5.6 percentage points, with improvements on all three held-out years. The resulting scaffold uses 8.40 calls per problem, compared with 3.51 for M1-SFT and 8.65 for Kimi-K3. With a similar number of calls to Kimi-K3, M1 exceeds its 40.0% accuracy, while GLM-5.3 achieves 48.1% with 9.39 calls. The SFT and RL results therefore offer different accuracy–call trade-offs: SFT yields an effective scaffold with fewer calls, while further RL finds a more accurate scaffold that uses more calls.

D.4 MLS-BENCH-LITE

We evaluate one task from each of the 12 domains in MLS-Bench-Lite. Table 12 lists the complete task set and the normalized rewards for Base and M1. Each model is evaluated once per task using the same Agentic MLE harness, task descriptions, interaction tools, and official Harbor task environments. Each task has a five-hour search budget and a maximum of 2,000 interaction steps, with a context limit of 200,000 tokens and an output limit of 65,536 tokens per step. CPU, GPU, memory, and verifier time limits follow each official task's resource specification.

The model edits the permitted algorithmic component, runs public experiments, and selects its final candidate through `update_answer`. Final performance is measured by the official hidden Harbor verifier, which aggregates task-specific metrics into a normalized reward. We report rewards on the zero-to-one scale, with higher values indicating better performance, and compute the mean reward as the arithmetic average over all 12 tasks. The mean rewards are 0.320514 for Base and 0.403192 for M1. M1 scores higher on ten tasks, ties on one, and scores lower on one.

Table 12: Results on 12 tasks selected from MLS-Bench-Lite, one per domain. Base and M1 are each evaluated once per task. Scores are normalized rewards from the official Harbor verifier; the final row is the arithmetic mean over the 12 tasks. Bold and underline mark the best and second-best values within each row, including ties.

Domain	Optimization task	Base	M1
Language Models	Diffusion language model demasking	0.148150	**0.386696**
Classical & Adaptive Learning	Clustering algorithm	0.439982	**0.525322**
Reinforcement Learning	Diffusion policy	0.129504	**0.161256**
Optimization & Theory	Multi-objective optimization	0.160624	**0.350473**
Robotics	Behavior cloning loss	0.435532	**0.444879**
Vision & Generation	3D Gaussian splatting densification	**0.000000**	**0.000000**
Deep Learning	Activation function	0.773039	**0.804121**
ML Systems & Efficient ML	Language model post-training quantization	0.443933	**0.456671**
AI for Science	Protein mutation effect prediction	0.509953	**0.518636**
Time Series & Forecasting	Exogenous time-series forecasting	**0.445408**	0.427787
Structured & Causal Reasoning	Discrete causal discovery	0.000000	**0.371316**
Trustworthy Learning	Membership inference defense	0.360039	**0.391151**
Average		0.320514	**0.403192**

Labels in Figure 4: LM, Language Models; Classical, Classical and Adaptive Learning; RL, Reinforcement Learning; Opt., Optimization and Theory; Robotics, Robotics; Vision, Vision and Generation; DL, Deep Learning; Systems, Machine Learning Systems and Efficient Machine Learning; AI4Sci, Artificial Intelligence for Science; TS, Time Series and Forecasting; Causal, Structured and Causal Reasoning; Trust., Trustworthy Learning.

Variation across optimization tasks. Table 12 shows that the largest absolute reward gains occur in causal discovery, language-model demasking, and multi-objective optimization. Other tasks show

smaller improvements, including behavior cloning, activation-function design, and protein mutation effect prediction. Both models receive zero reward on Gaussian splatting densification, while M1 scores lower on time-series forecasting. The aggregate improvement thus reflects gains across several algorithmic components, with task-dependent benefits under the shared search budget.