ACEGRPO: ADAPTIVE CURRICULUM ENHANCED GROUP RELATIVE POLICY OPTIMIZATION FOR AUTONOMOUS MACHINE LEARNING ENGINEERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Autonomous Machine Learning Engineering (MLE) requires agents to perform sustained, iterative optimization over long horizons. While recent LLM-based agents show promise, current prompt-based agents for MLE suffer from behavioral stagnation due to frozen parameters. Although Reinforcement Learning (RL) offers a remedy, applying it to MLE is hindered by prohibitive execution latency and inefficient data selection. Recognizing these challenges, we propose **AceGRPO** with two core components: (1) **Evolving Data Buffer** that continuously repurposes execution traces into reusable training tasks, and (2) **Adaptive Sampling** guided by a Learnability Potential function, which dynamically prioritizes tasks at the agent's learning frontier to maximize learning efficiency. Trained with AceGRPO, **Ace-35B** substantially improves over its base model, achieving a **100% valid submission rate** on MLE-Bench-Lite and approaching or surpassing larger frontier models on key metrics. These results demonstrate the effectiveness of adaptive, execution-grounded training for building capable and reliable long-horizon MLE agents.

1 INTRODUCTION

Large Language Models (LLMs) have transitioned from static question-answering systems into autonomous agents capable of complex reasoning, tool use, and multi-step problem solving (Tang et al., 2025; Wang et al., 2024; Zhang et al., 2025). This transition has enabled LLM agents to tackle tasks that demand not single-shot answers but sustained iterative optimization. Autonomous Machine Learning Engineering (Autonomous MLE)—including settings such as Kaggle competitions (Chan et al., 2024)—exemplifies this class of challenges. Unlike standard Software Engineering where success is often binary (e.g., passing unit tests) (Jimenez et al., 2023; Du et al., 2025b), Autonomous MLE constitutes an empirical science requiring continuous refinement: agents must search high-dimensional hypothesis spaces, propose changes to architectures or data pipelines, and interpret noisy experimental feedback to improve performance over time (Liu et al., 2025a). Progress in this domain necessitates more than coding proficiency; it demands *self-evolution*—the ability to sustain improvement trajectories, escape local optima, and refine strategies based on accumulated experience.

Most recent progress on Autonomous MLE relies on prompt-based methods (Yang et al., 2025b; Li et al., 2025; Nam et al., 2025) to structure agent exploration at test time. While these methods scale inference-time search, they leave the underlying LLM parameters fixed. Because the policy does not update, the agent cannot convert trial-and-error experience into improved decision rules, and the policy stagnates. As a result, agents repeatedly explore suboptimal patterns even after thousands of episodes, yielding persistent behavioral plateaus. A natural alternative is to shift from *searching* for solutions to *training* better solvers using reinforcement learning (RL) (Lu et al., 2025). Such a transition is essential to empower LLMs to sustain self-evolution and master the dynamics of iterative optimization in long-horizon MLE tasks (Zhu et al., 2026). Leveraging the objective and verifiable metrics provided by MLE tasks (Qiang et al., 2025), RL serves as a powerful method to steer the optimization process (Guo et al., 2025).

However, establishing RL training for MLE remains an open challenge. Success in MLE necessitates sustained iterative refinement unlike one-shot coding tasks (Huang et al., 2023). Training for

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.

such capabilities is hindered by data that is scarce and unevenly distributed (Rengarajan et al., 2022). The challenge is further compounded by the prohibitive latency of execution, where a single step may require minutes to hours, rendering end-to-end full-trajectory RL computationally intractable (Liu et al., 2025b; Deng et al., 2025). Consequently, practical approaches must adopt step-wise rollouts, optimizing single actions conditioned on intermediate states to ensure training efficiency. Nevertheless, step-wise RL in Autonomous MLE faces two challenges: (1) *Where to sample states*: prior approaches often construct a large state pool by collecting trajectories with varying iteration depths which incurs substantial upfront cost and typically yields limited diversity (Deng et al., 2025). (2) *How to sample states*: standard sampling (e.g., uniform sampling) over the pool frequently selects states that are either mastered (deterministic high rewards) or beyond capability (deterministic failures) which leads to vanishing within-group reward dispersion, yielding ineffective updates and wastes computation (Chen et al., 2026).

To address these challenges, we propose **AceGRPO** (**A**daptive **C**urriculum **E**nhanced **GRPO**), an RL training framework that bootstraps an evolving pool of intermediate states to enable continuous, open-ended self-improvement. AceGRPO is built upon two tightly-coupled components: (1) To balance pool diversity with collection efficiency, we introduce an **Evolving Data Buffer** that continuously converts expensive executions into reusable step-wise training tasks. It treats every intermediate state (whether from failed debugging attempts or suboptimal solutions) as a valid starting point for a new single-step RL task. (2) To spend execution budget on informative states, we design an **Adaptive Sampling** strategy that prioritizes states exhibiting both high reward variance (indicating uncertainty near the agent's capability frontier) and meaningful improvement headroom (avoiding mastered tasks with diminishing returns). By strategically sampling these high-value starting points, AceGRPO constructs an adaptive curriculum that concentrates computation on the agent's evolving Learning Zone, preserving exploration diversity while improving training efficiency.

We train **Ace-35B** from Qwen3.6-35B-A3B (Qwen Team, 2026a) using AceGRPO. On MLE-Bench-Lite (Chan et al., 2024), Ace-35B reaches a **100% valid submission rate**, a **62.12% Any Medal rate**, and a **77.14 HumanRank score**. It substantially improves over its base model by 13.64 percentage points in Any Medal rate and 10.85 HumanRank points. These gains bring Ace-35B close to larger frontier models and ahead of them on key metrics: it exceeds Qwen3.8-Max in both Any Medal and HumanRank, comes within 1.43 HumanRank points of DeepSeek-V4-Pro, and achieves a higher valid submission rate than every evaluated GLM, DeepSeek, and Kimi baseline. These results demonstrate that adaptive, execution-grounded training can substantially strengthen a compact base model and make it competitive with larger frontier systems. Our code is available at https://anonymous.4open.science/r/AceGRPO.

Our contributions are summarized as follows:

- We propose **AceGRPO**, an adaptive RL framework that reformulates long-horizon MLE optimization as step-wise learning over an **Evolving Data Buffer**, enabling continuous self-evolution.
- We introduce **Adaptive Sampling** guided by **Learnability Potential**. By serving as a proxy for gradient magnitude, this mechanism dynamically prioritizes tasks at the agent's learning frontier, ensuring exploration diversity while maximizing training efficiency.
- On **MLE-Bench-Lite**, Ace-35B substantially improves over its base model and approaches or surpasses larger frontier models on key metrics, achieving a **62.12% Any Medal rate** with **100% valid submissions**.

2 Related Work

2.1 Autonomous Machine Learning Engineering

The evolution of LLM-driven agents facilitates autonomous Machine Learning Engineering (MLE), shifting focus from static QA to complex, iterative optimization. Foundational benchmarks like MLE-Bench Chan et al. (2024) and platforms like OpenHands Wang et al. (2024) established the necessity for tight execution feedback loops. To navigate extended horizons, contemporary approaches predominantly rely on inference-time exploration strategies. Methods such as AIDE Jiang et al. (2025), R&D-Agent Yang et al. (2025b), and ML-Master Liu et al. (2025a) utilize prompt-based planning, while AIRA Toledo et al. (2025), AutoMLGen Du et al. (2025a), and FM Agent Li et al. (2025) incorporate evolutionary mechanisms for cross-branch knowledge sharing; others focus on tool

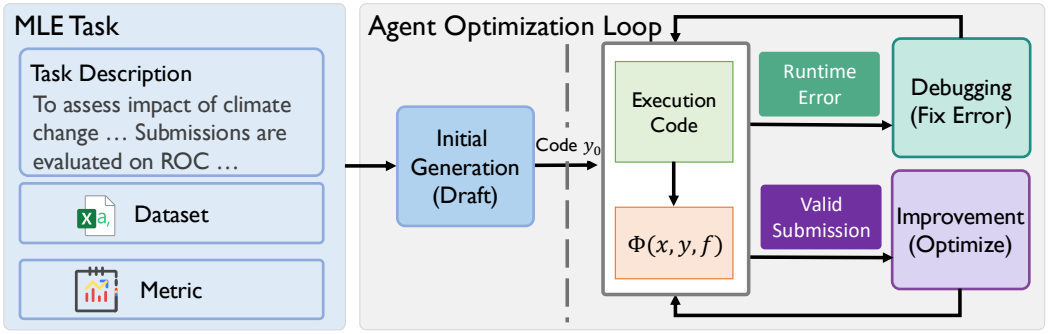

Figure 1: The Agent Optimization Loop for MLE. It is a continuous iteration over three distinct phases: Draft (initial generation), Debug (error correction), and Improve (metric optimization). The transitions are deterministically governed by the execution feedback, allowing the agent to refine its code solution and accumulate context through repeated interactions.

optimization Sahney et al. (2025), targeted initialization Nam et al. (2025) or context management Zhu et al. (2026). However, these parameter-frozen approaches lack mechanisms to internalize learned strategies, leading to behavioral stagnation. While Reinforcement Learning (RL) offers a pathway to train capable solvers, applying it to long-horizon MLE is hindered by the prohibitive latency of full-trajectory feedback and the limited exploration offered by static datasets. Our work resolves this tension with AceGRPO, which bootstraps a dynamic task pool using a Learnability Potential to prioritize high-value intermediate states, ensuring sample-efficient self-evolution.

2.2 Agentic Reinforcement Learning for LLMs

The emergence of Agentic Reinforcement Learning (Agentic RL) marks a fundamental shift from treating LLMs as static text emitters to conceptualizing them as learnable policies embedded within sequential decision loops Cao et al. (2024). This paradigm has catalyzed advances in "slow reasoning" capabilities, where models like DeepSeek-R1 Guo et al. (2025) and o1-Coder Zhang et al. (2024) internalize deliberate thought processes and self-correction via reinforcement signals. In the domain of software engineering, AceCoder Zeng et al. (2025) and DeepCoder Luo et al. (2025b) progress from optimizing single-turn code generation to mastering complex, repository-scale workflows. Systems such as DeepSWE Luo et al. (2025a), SWE-RL Wei et al. (2025), and Qwen3-Coder Yang et al. (2025a) demonstrate that RL with execution-based feedback can effectively optimize agents for long-horizon debugging and feature implementation. Within MLE domains, ML-Agent Liu et al. (2025b) extends this efficacy to automated machine learning pipelines. However, applying these methods to MLE remains challenged by the tension between the high latency of full-trajectory rollouts and the limited exploration of static datasets Zhao et al. (2025); Golubev et al. (2025). Our work addresses this bottleneck via AceGRPO, which employs an adaptive curriculum to dynamically prioritize high-potential intermediate states, ensuring efficient self-evolution without the computational prohibitive cost of redundant long-horizon exploration.

3 Problem Formulation

We formalize Machine Learning Engineering (MLE) as an iterative code optimization process. While the interaction naturally unfolds as a long-horizon agent-environment loop, we reformulate the learning objective as step-wise optimization over a dynamically evolving task distribution to address the fundamental efficiency constraints.

3.1 MLE as Iterative Optimization with Task Expansion

An MLE task instance is defined as a tuple $I = (\iota, \mathcal{D}, m)$, where ι is a natural-language task description, $\mathcal{D}$ represents the dataset, and $m : \mathcal{Y} \to \mathbb{R}$ is a metric function evaluating code $y \in \mathcal{Y}$.

The agent operates on a state (context) $x = (I, c, h) \in \mathcal{X}$, comprising the task instance I, the current code snapshot c, and an execution history $h = \{(y_k, f_k, s_k)\}_{k=0}^{|h|}$. Here, each entry records a past solution y_k, its execution feedback f_k, and the corresponding evaluation score s_k derived from the

metric m (or a failure indicator if execution fails). Given a context x, the policy $\pi_\theta(\cdot \mid x)$ generates a code update y. Depending on the state of c and h, this action corresponds to three distinct operational phases, as illustrated in fig. 1: *Initial Generation* (when $c = \emptyset$), *Debugging* (when the latest feedback in h indicates failure), or *Improvement* (when c is valid but suboptimal). Executing y against the environment yields feedback $f = \mathrm{Exec}(x, y)$ and a scalar reward $r = \mathcal{R}(x, y)$.

Crucially, rather than treating the interaction as a static dataset, we model it as a dynamic expansion process. Each execution deterministically spawns a new derivative state via a transition operator Φ:

$$x' = \Phi(x, y, f) = (I, y, h \cup \{(y, f, r)\}). \tag{1}$$

Let $\mathcal{B}_t \subset \mathcal{X}$ denote the evolving context buffer at iteration t. The buffer expands recursively as $\mathcal{B}_{t+1} \leftarrow \mathcal{B}_t \cup \{\Phi(x, y, f)\}$. Since optimizing the full long-horizon trajectory return is intractable due to sparse feedback and high latency, we actuate a step-wise optimization as a substitute. We train the policy to maximize the expected immediate reward over an adaptive sampling distribution $\mathcal{Q}_t$ defined on the current buffer $\mathcal{B}_t$:

$$J_t(\theta) = \mathbb{E}_{x \sim \mathcal{Q}_t}\left[\mathbb{E}_{y \sim \pi_\theta(\cdot|x)}\left[\mathcal{R}(x, y)\right]\right]. \tag{2}$$

This formulation reframes MLE from standard trajectory optimization into *active context selection*: the core challenge shifts to designing $\mathcal{Q}_t$ to strictly prioritize states where the gradient signal maximizes the agent's self-evolution per unit of execution cost.

3.2 Reinforcement Learning for LLMs

For a sampled context $x \sim \mathcal{Q}_t$, GRPO draws a group $\{y_i\}_{i=1}^G$ from $\pi_{\theta_{\text{old}}}(\cdot \mid x)$ and evaluates $r_i = \mathcal{R}(x, y_i)$. Define

$$\mu(x) = \frac{1}{G}\sum_{i=1}^G r_i, \quad \sigma(x) = \sqrt{\frac{1}{G}\sum_{i=1}^G (r_i - \mu(x))^2}, \quad \hat{A}_i(x) = \frac{r_i - \mu(x)}{\sigma(x) + \varepsilon}. \tag{3}$$

Let $\rho_i(\theta) = \pi_\theta(y_i \mid x)/\pi_{\theta_{\text{old}}}(y_i \mid x)$. With clipping δ and KL regularization to a reference policy π_{ref} weighted by β, GRPO maximizes

$$\begin{aligned}\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_{x \sim \mathcal{Q}_t}\Big[\frac{1}{G}\sum_{i=1}^G \min\Big(\rho_i(\theta)\hat{A}_i(x),\ \mathrm{clip}(\rho_i(\theta), 1-\delta, 1+\delta)\hat{A}_i(x)\Big) \\ - \beta\,\mathrm{KL}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x))\Big].\end{aligned} \tag{4}$$

3.3 Challenges in RL for Dynamic MLE Tasks

Prohibitive feedback latency. Unlike fast-verification domains, evaluating $\mathcal{R}(x, y)$ often requires running full training MLE pipelines, making multi-step online trajectory collection impractical at scale. This motivates the step-wise objective in Eq. (2) and shifts the bottleneck to selecting informative contexts x under a fixed execution budget.

Non-stationary expanding pools and variance collapse. Because $\mathcal{T}_t$ grows via Φ, naive sampling (e.g., uniform over $\mathcal{T}_t$) is dominated by contexts that quickly become either (i) *mastered*, where $r_i \approx r_{\max}$ for all i, or (ii) *beyond-frontier*, where $r_i \approx r_{\min}$ for all i. In both regimes, $r_i - \mu(x) \to 0$, hence $\hat{A}_i(x) \to 0$, and GRPO update loses signal:

$$\nabla_\theta \mathcal{L}_{\text{GRPO}}(\theta) \ \propto\ \mathbb{E}\big[\nabla_\theta \log \pi_\theta(y_i \mid x)\ \hat{A}_i(x)\big] \ \approx\ 0. \tag{5}$$

Therefore, uniform sampling wastes compute on "empty" steps. This motivates **AceGRPO**, which replaces naive $\mathcal{Q}_t$ with an adaptive curriculum that targets contexts in the agent's learning zone.

4 AceGRPO

To address the prohibitive execution latency and the stage-varying ability requirements in long-horizon Machine Learning Engineering (MLE), we propose **AceGRPO**, an RL framework built upon two tightly-coupled components: (1) an **Evolving Data Buffer** that continuously converts expensive executions into reusable step-wise training tasks, and (2) an **Adaptive Sampling** strategy

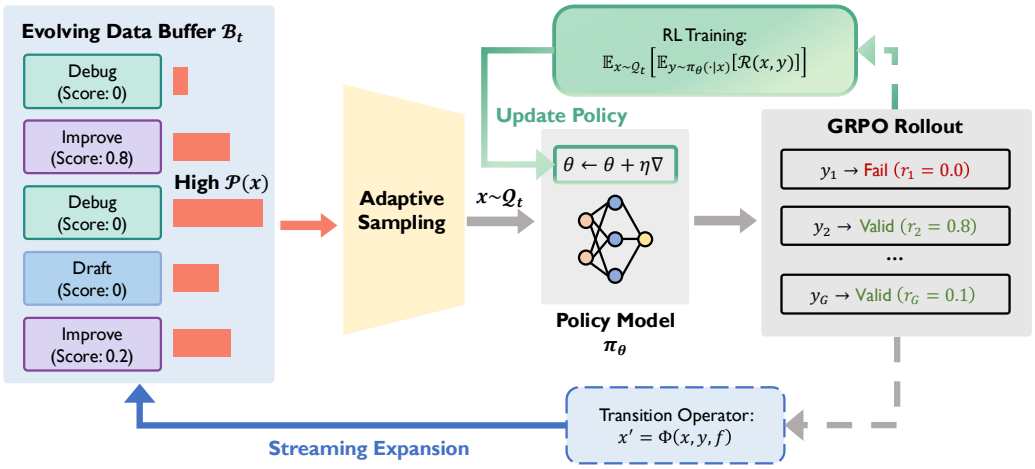

Figure 2: The AceGRPO framework. MLE is step-wise optimization over a dynamic task distribution. The method features an Evolving Data Buffer that accumulates intermediate states (Draft, Debug, Improve) via Streaming Expansion. An Adaptive Sampler selects high-potential tasks to maximize the gradient signal for GRPO training, enabling efficient self-improvement on long-horizon challenges. that adaptively allocates the limited execution budget to the agent's current learning frontier and the capability emphasized at the current optimization stage.

4.1 Evolving Data Buffer: Streaming Construction of Step-wise Training Tasks

While MLE interactions naturally unfold as long trajectories, optimizing them via full-trajectory RL is intractable due to high feedback latency and prohibitive execution costs. AceGRPO operationalizes the step-wise objective in Equation (2) by maintaining an **evolving data buffer** $\mathcal{B}_t$. This buffer serves as a dynamic context pool, allowing the agent to continuously repurpose expensive execution traces into distinct, replayable optimization steps. In practice, the outer-loop sampling distribution $\mathcal{Q}_t$ is defined directly over this buffer, i.e., $\mathrm{supp}(\mathcal{Q}_t) \subseteq \mathcal{B}_t$.

Buffer content. Each item in $\mathcal{B}_t$ represents an intermediate state $x = (I, c, h)$, fully encapsulating the problem description, current code snapshot, and execution history. To facilitate adaptive sampling, we augment each state with lightweight metadata: (i) a *task type* $\kappa(x) \in \{\textsc{Draft}, \textsc{Debug}, \textsc{Improve}\}$, and (ii) a *learnability potential* $\mathcal{P}(x)$ (detailed in Section 4.2). Here, Draft denotes solving from the task description, Debug denotes repairing an invalid execution using error feedback, and Improve denotes refining a valid solution using its previous score as a baseline. The $\mathcal{P}(x)$ is initialized to a default value $\mathcal{P}_{\text{init}}$ upon insertion and it is updated only when x is sampled from the buffer and used as the initial state of a new rollout. In this case, $\mathcal{P}(x)$ is updated using the reward produced by the most recent rollout.

Streaming expansion. At training step t, we sample a batch of starting states $x \sim \mathcal{Q}_t(\mathcal{B}_t)$ and generate a group of G candidate actions $\{y_i\}_{i=1}^{G} \sim \pi_\theta(\cdot \mid x)$. Executing each y_i yields feedback f_i (e.g., error logs or leaderboards) and a raw execution score s_i. Crucially, strictly following the transition operator defined in Section 3, each execution spawns a new derivative state $x'_i = \Phi(x, y_i, f_i)$, which is immediately appended to $\mathcal{B}_t$. The task type $\kappa(x'_i)$ is assigned logically based on the outcome: failed runs generate Debug states enriched with tracebacks, while successful runs produce Improve states containing new score as a baseline.

This mechanism transforms every execution into two distinct assets: a gradient signal for the current policy update and a novel starting point for future curriculum learning. By expanding $\mathcal{B}_t$ on-the-fly with the agent's own outputs, AceGRPO ensures that the training distribution continuously shifts towards the agent's evolving frontier, preserving fresh policy-generated diversity without the cost of redundant full-trajectory rollouts.

Reward shaping. We adopt the *HumanRank score* (Qiang et al., 2025) as the unified metric. Let p be the rank of a submission among N human participants. The normalized score is $s = 1 - \frac{p}{N}$. Invalid submissions are assigned $s = -1$. We define the immediate reward $R(x, y)$ as a mixture of absolute performance and relative improvement over the state's previous baseline $s_p(x)$. For Draft

and DEBUG tasks, $s_p(x) = 0$; for IMPROVE tasks, $s_p(x)$ is the previous valid score inherited from the parent state.

$$R(x, y) = \begin{cases} 0, & s = -1, \\ (1-\alpha)s + \alpha \max\left(0, \dfrac{s - s_{\mathrm{p}}(x)}{1 - s_{\mathrm{p}}(x) + \epsilon}\right), & \text{otherwise}, \end{cases} \tag{6}$$

where $\alpha \in [0, 1]$ balances the terms and ϵ ensures numerical stability. This shaping strategy creates a stage-consistent incentive: early training focuses on achieving valid solutions (via the absolute term s), while later stages are explicitly driven to exploit the diminishing improvement headroom (via the relative term).

Updating potential from GRPO group outcomes. When a state x is sampled, the GRPO rollout produces a reward set $\mathbf{r}_x^{(t)} = \{R(x, y_1), \ldots, R(x, y_G)\}$. We utilize the within-group statistics of $\mathbf{r}_x^{(t)}$ to update $\mathcal{P}(x)$, as this variance serves as a direct proxy for the learning signal's magnitude (see Section 4.2). For any newly spawned state x', we set $\mathcal{P}(x') = \mathcal{P}_{\text{init}}$ pending its first execution.

4.2 ADAPTIVE SAMPLING: DYNAMIC PRIORITIZATION VIA LEARNABILITY POTENTIAL

The buffer $\mathcal{B}_t$ provides a rich and expanding substrate of training states. However, each sampled starting state triggers expensive executions, and the effectiveness of a GRPO update depends critically on whether the sampled state induces non-trivial within-group reward variation. Standard uniform sampling is inefficient because the buffer inevitably accumulates states that are either *mastered* (yielding deterministic high rewards) or *beyond capability* (yielding deterministic failures). Both regimes produce vanishing within-group advantage variance ($\hat{A}_i \approx 0$), resulting in wasted computation. Consequently, we explicitly design the outer-loop sampling distribution $\mathcal{Q}_t$ (Equation (2)) as a strategic resource allocation mechanism for allocating execution budget.

We construct $\mathcal{Q}_t$ as a potential-weighted distribution over task types, combined with lightweight diversity terms that preserve long-term coverage.

Learnability Potential as a Gradient Proxy. We introduce *Learnability Potential*, $\mathcal{P}(x)$, to quantify the expected informativeness of a state based on its most recent execution outcomes. Since the magnitude of the GRPO update is proportional to the advantage variance, the dispersion of $\mathbf{r}_x^{(t)}$ serves as a direct proxy for the learning signal. We define $\mathcal{P}(x)$ as:

$$\mathcal{P}(x) = \underbrace{\mathrm{clip}\big(\sigma(\mathbf{r}_x^{(t)}), 0, \delta_{\max}\big)}_{\text{Uncertainty}} \cdot \underbrace{\mathrm{clip}\big(1 - \mu(\mathbf{r}_x^{(t)}), 0, 1\big)}_{\text{Headroom}}, \tag{7}$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ denote the mean and standard deviation, respectively. Here, $\delta_{\max}$ caps extreme variance to prevent instability. The *Uncertainty* term targets states in the "learning zone" where outcomes are non-deterministic and within-group variance is non-zero, mitigating GRPO gradient collapse. The *Headroom* term only suppresses saturated high-mean states; it is not intended to penalize low-mean states by itself. Instead, the multiplicative form makes uncertainty a gate on headroom: deterministic failures receive low potential because $\sigma(\mathbf{r}_x^{(t)}) \approx 0$, while states receive high potential only when they exhibit both non-trivial reward dispersion and remaining improvement space. For any newly inserted state x' not yet executed, we initialize $\mathcal{P}(x') = \mathcal{P}_{\text{init}}$.

Smooth curriculum with recency. Directly sampling proportional to $\mathcal{P}(x)$ can lead to distribution collapse, where the agent overfits to a narrow set of high-variance tasks. To ensure robustness, AceGRPO uses a temperature-controlled sampler that is exploratory when the buffer is small and becomes more selective as the buffer grows:

$$\tau_t = \tau_0 \max\left(\epsilon_\tau, 1 - \frac{|\mathcal{B}_t|}{N_{\text{sat}}}\right) \tag{8}$$

We also down-weight states that were sampled very recently:

$$D_t(x) = \min\left(1, \frac{t - t_{\text{last}}(x)}{\Delta}\right), \tag{9}$$

where $t_{\text{last}}(x)$ is the last step when x was selected. This simple recency factor plays the same role as a refractory period, preventing a few states from monopolizing the rollout budget.

Table 1: Main results on MLE-Bench-Lite (mean ± standard deviation). Blocks list proprietary models, larger open-model baselines, Qwen3.8-Max, and the Qwen3.6-35B-A3B/Ace-35B pair. Bold highlights the trained Ace-35B model. All metrics are percentages except HumanRank, which is multiplied by 100.

Model	Valid Submission	Above Median	Bronze	Silver	Gold	Any Medal	HumanRank Score
Claude-4.5-Sonnet	100.00 ± 0.00	80.30 ± 2.14	7.58 ± 5.67	16.67 ± 5.67	36.36 ± 3.71	60.61 ± 5.67	75.86 ± 1.88
Gemini-3-Pro	100.00 ± 0.00	84.85 ± 4.29	4.55 ± 3.71	15.15 ± 4.29	39.39 ± 2.14	59.09 ± 3.71	79.18 ± 2.40
GPT-5.2	95.45 ± 3.71	71.21 ± 4.29	10.61 ± 4.29	16.67 ± 2.14	28.79 ± 2.14	56.06 ± 4.29	71.05 ± 0.91
GLM-5.3	93.94 ± 4.29	84.85 ± 4.29	3.03 ± 2.14	22.73 ± 3.71	46.97 ± 2.14	72.73 ± 3.71	81.36 ± 3.42
GLM-5.3-Flash	98.48 ± 2.14	86.36 ± 0.00	6.06 ± 2.14	18.18 ± 3.71	43.94 ± 2.14	68.18 ± 3.71	82.32 ± 1.24
DeepSeek-V4-Pro-0813	98.48 ± 2.14	83.33 ± 2.14	10.61 ± 2.14	18.18 ± 3.71	37.88 ± 2.14	66.67 ± 2.14	78.57 ± 1.50
DeepSeek-V4.1-Flash	96.97 ± 2.14	83.33 ± 2.14	10.61 ± 4.29	13.64 ± 6.43	43.94 ± 7.73	68.18 ± 3.71	80.51 ± 2.65
Kimi-K2.7-Code	98.48 ± 2.14	77.27 ± 3.71	10.61 ± 4.29	18.18 ± 0.00	34.85 ± 2.14	63.64 ± 6.43	75.30 ± 1.98
Qwen3.8-Max	92.42 ± 4.29	75.76 ± 2.14	7.58 ± 2.14	13.64 ± 0.00	30.30 ± 2.14	51.52 ± 2.14	71.06 ± 3.52
Qwen3.6-35B-A3B	96.97 ± 2.14	65.15 ± 2.14	1.52 ± 2.14	16.67 ± 4.29	30.30 ± 2.14	48.48 ± 2.14	66.29 ± 0.95
Ace-35B	**100.00 ± 0.00**	**77.27 ± 3.71**	**4.55 ± 3.71**	**21.21 ± 2.14**	**36.36 ± 3.71**	**62.12 ± 5.67**	**77.14 ± 4.08**

Final sampling distribution. Combining the potential-based priority, temperature, and recency factor, the final probability of sampling state x at iteration t is given by:

$$\mathcal{Q}_t(x) = \frac{1}{Z_t} \exp\left(\frac{\mathcal{P}(x)}{\tau_t}\right) D_t(x), \tag{10}$$

where Z_t is the partition function normalizing the distribution over $\mathcal{B}_t$. This construction dynamically aligns the execution budget with the agent's evolving competency while preserving necessary exploration diversity.

4.3 Asynchronous Training Architecture

To handle the latency of MLE, we implement AceGRPO using a decoupled, asynchronous architecture. The system separates **Rollout Workers** from **Learner Actors**. Workers continuously sample starting states $x \sim \mathcal{Q}_t$ from the evolving buffer $\mathcal{B}_t$, execute actions, and asynchronously perform the streaming expansion of $\mathcal{B}_t$ with new states. Simultaneously, Learner Actors consume the collected interaction data to perform GRPO updates. This parallelization refreshes curriculum statistics and buffer content in real time without blocking optimization, effectively maximizing the throughput of valid learning samples.

5 Experiments

5.1 Experimental Setup

Training settings. We initialize our policy from Qwen3.6-35B-A3B (Qwen Team, 2026a) and train it with AceGRPO to obtain Ace-35B. We use MLE-Dojo (Qiang et al., 2025) to construct our training dataset, removing the 68 tasks that overlap with MLE-Bench (Chan et al., 2024), leaving 134 machine learning competition tasks. Training runs for 400 steps with 8 tasks per rollout batch, 8 responses per task, and a global training batch size of 64. Additional settings and prompts are provided in Appendices A.1 and E.

Evaluation settings. We evaluate on MLE-Bench-Lite, a 22-task subset of MLE-Bench (Chan et al., 2024), with a 12-hour budget per task. We report valid submission, above-median, Bronze, Silver, Gold, and Any Medal rates, together with the HumanRank score from MLE-Dojo (Qiang et al., 2025)Details about metrics and MLE-Bench are provided in A.2. Each reported evaluation is repeated three times, and Table 1 gives means and standard deviations. We compare our approach against several strong baselines: (1) proprietary models Claude-4.5-Sonnet (Anthropic, 2025), Gemini-3-Pro (Pichai et al., 2025), and GPT-5.2 (OpenAI, 2025); (2) larger open-model baselines GLM-5.3 (Z.ai, 2026a), GLM-5.3-Flash (Z.ai, 2026b), DeepSeek-V4-Pro-0813 (DeepSeek, 2026b), DeepSeek-V4.1-Flash (DeepSeek, 2026a), and Kimi-K2.7-Code (Moonshot AI, 2026); (3) Qwen3.8-Max (Qwen

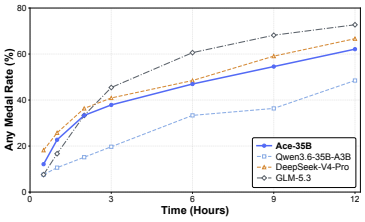

Figure 3: Mean medal rate over time for four selected models.

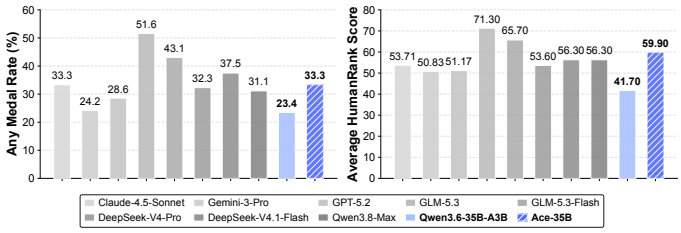

Figure 4: First-valid-submission performance: medal rate (left) and mean HumanRank on the 0–100 scale (right), computed over trials with a valid submission.

Team, 2026b); and (4) Qwen3.6-35B-A3B (Qwen Team, 2026a) and our Ace-35B. All comparisons below refer to the reported means.

5.2 Main Results

AceGRPO is an effective reinforcement learning method that substantially improves end-to-end MLE performance. Table 1 presents a comprehensive comparison with proprietary frontier and large open-source LLMs on the MLE-Bench-Lite. As it shows, Ace-35B consistently outperforms the baseline Qwen3.6-35B-A3B in all major evaluation metrics. In particular, the Any Medal rate increases by 13.64% from 48.48% to 62.12%, and the HumanRank score improves 10.85 from 66.29 to 77.14. Notably, Ace-35B also achieves a **100% Valid Submission rate**, matching the strongest proprietary LLMs and indicating substantially improved robustness in long-horizon MLE tasks. These consistent gains across multiple metrics demonstrate that AceGRPO yields effective reinforcement learning updates, enabling LLMs to internalize machine learning engineering strategies beyond inference-time prompting.

Ace-35B approaches or surpasses larger frontier LLMs on key metrics. As shown in Table 1, Ace-35B competes with larger open models, including GLM-5.3-Flash (320B total parameters) (Z.ai, 2026b) and DeepSeek-V4.1-Flash (552B) (DeepSeek, 2026a). Ace-35B narrows the HumanRank gap to DeepSeek-V4-Pro from 12.28 to 1.43 (77.14 versus 78.57) and the Any Medal gap from 18.19% to 4.55%. Its **100% valid submission rate** exceeds all evaluated GLM, DeepSeek, and Kimi baselines (93.94%–98.48%). It surpasses Qwen3.8-Max by 10.60% in Any Medal and 6.08 HumanRank. Against Kimi-K2.7-Code, Ace-35B achieves a 1.84 higher HumanRank score, with an Any Medal rate 1.52% points lower. Its mean Any Medal rate exceeds Claude-4.5-Sonnet, Gemini-3-Pro, and GPT-5.2; its HumanRank exceeds Claude and GPT-5.2 and comes within 2.04 points of Gemini-3-Pro. These results show that task-specific training brings Ace-35B close to larger frontier models, with advantages on key MLE metrics.

AceGRPO enables sustained self-evolution rather than one-shot performance gains. Figure 3 compares Ace-35B with its base model, DeepSeek-V4-Pro, and GLM-5.3; all seven models are shown in Appendix F. Whereas Qwen3.6-35B-A3B largely plateaus within the first six hours, Ace-35B continues to improve steadily throughout the entire run, consistently surpassing the base model and progressively narrowing the gap to Deepseek-v4-pro and GLM-5.3. This persistent upward trajectory suggests that AceGRPO facilitates sustained self-improvement over extended horizons, rather than yielding isolated or brittle gains, supporting its effectiveness for MLE-style tasks.

5.3 Analysis

In this section, we analyze Ace-35B during training, over the evaluation horizon, and at the first valid submission. Ablation studies are provided in Appendix B.

AceGRPO exhibits consistent and effective training dynamics. Figure 5 illustrates the training dynamics of AceGRPO. Figure 5 (Left) depicts the evolution of the valid submission rate for the Draft, Debug, and Improve tasks. The consistent upward trend across all three tasks suggests steady gains in the LLM's baseline capabilities. Figure 5 (Middle) presents the reward trajectories for these tasks throughout the training process. The concurrent rise in rewards demonstrates the model's enhanced

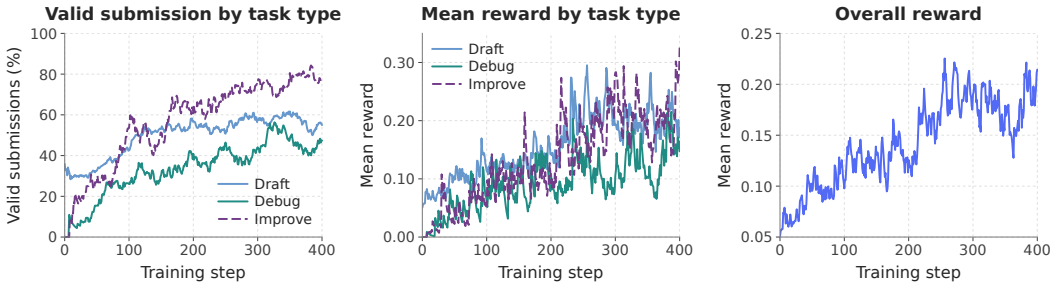

Figure 5: Training dynamics of Ace-35B over 400 steps: valid submission rates and rewards across task types.

proficiency in generating high-quality code and solving machine learning problems. Figure 5 (Right) illustrates the overall reward curve aggregated across all tasks, which exhibits a increase. Taken together, these results indicate that the LLM improves continuously during training, as evidenced by rising valid submission rates and rewards across all task types.

First-submission quality. Figure 4 shows that Ace-35B improves the first-valid-submission medal rate from 23.44% to 33.33% and the mean HumanRank score from 41.70 to 59.90 relative to Qwen3.6-35B-A3B. Ace-35B also achieves higher first-submission medal rates and mean HumanRank scores than DeepSeek-V4-Pro and Qwen3.8-Max, and a higher mean HumanRank score than DeepSeek-V4.1-Flash. These results demonstrate that AceGRPO strengthens solution quality at the first valid submission, enabling the trained model to approach or surpass larger frontier models on these metrics.

6 Conclusion

In this work, we presented **AceGRPO**, a reinforcement learning framework for long-horizon autonomous Machine Learning Engineering that turns costly trajectory-level feedback into step-wise optimization over an evolving task distribution. Its Evolving Data Buffer reuses intermediate execution states, while Adaptive Sampling prioritizes states with informative learning signals.

Training Qwen3.6-35B-A3B with AceGRPO for 400 steps yields Ace-35B, which substantially improves over its base model: Any Medal increases by 13.64 percentage points, HumanRank rises by 10.85 points, and valid submissions reach 100%. These gains bring Ace-35B close to larger frontier models and ahead of them on key metrics, including stronger Any Medal and HumanRank results than Qwen3.8-Max and higher valid submission rates than the evaluated GLM, DeepSeek, and Kimi baselines. Overall, the results demonstrate that adaptive, execution-grounded training can substantially strengthen a compact model and make it competitive with larger frontier systems for long-horizon MLE.

7 Limitations

Due to the high execution cost of long-horizon MLE environments, our experiments are conducted under a constrained training budget and limited rollout scale. Larger-scale training and longer curriculum evolution may further improve performance, but were not explored in this work.

AI use statement

We used generative AI tools to assist with language polishing and to improve the readability and fluency of the manuscript.

References

Anthropic. Introducing Claude Sonnet 4.5, September 2025. URL https://www.anthropic.com/news/claude-sonnet-4-5.

Yuji Cao, Huan Zhao, Yuheng Cheng, Ting Shu, Yue Chen, Guolong Liu, Gaoqi Liang, Junhua Zhao, Jinyue Yan, and Yun Li. Survey on large language model-enhanced reinforcement learning: Concept, taxonomy, and methods. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Xiwen Chen, Wenhui Zhu, Peijie Qiu, Xuanzhao Dong, Hao Wang, Haiyu Wu, Huayu Li, Aristeidis Sotiras, Yalin Wang, and Abolfazl Razi. Dra-grpo: Your grpo needs to know diverse reasoning paths for mathematical reasoning, 2026. URL https://arxiv.org/abs/2505.09655.

DeepSeek. DeepSeek-V4.1-Flash: Smarter, faster, more efficient, September 2026a. URL https://api-docs.deepseek.com/news/news260910/.

DeepSeek. DeepSeek-V4-Pro GA release, August 2026b. URL https://api-docs.deepseek.com/news/news260813/.

Yihe Deng, I Hsu, Jun Yan, Zifeng Wang, Rujun Han, Gufeng Zhang, Yanfei Chen, Wei Wang, Tomas Pfister, Chen-Yu Lee, et al. Supervised reinforcement learning: From expert trajectories to step-wise reasoning. *arXiv preprint arXiv:2510.25992*, 2025.

Shangheng Du, Xiangchao Yan, Dengyang Jiang, Jiakang Yuan, Yusong Hu, Xin Li, Liang He, Bo Zhang, and Lei Bai. Automlgen: Navigating fine-grained optimization for coding agents. *arXiv preprint arXiv:2510.08511*, 2025a.

Yaxin Du, Yuzhu Cai, Yifan Zhou, Cheng Wang, Yu Qian, Xianghe Pang, Qian Liu, Yue Hu, and Siheng Chen. Swe-dev: Evaluating and training autonomous feature-driven software development. *arXiv preprint arXiv:2505.16975*, 2025b.

Alexander Golubev, Maria Trofimova, Sergei Polezhaev, Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Sergey Abramov, Andrei Andriushchenko, Filipp Fisin, et al. Training long-context, multi-turn software engineering agents with reinforcement learning. *arXiv preprint arXiv:2508.03501*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.

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.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

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.

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Siheng Chen, et al. Ml-master: Towards ai-for-ai via integration of exploration and reasoning. arXiv preprint arXiv:2506.16499, 2025a.

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*, 2025b.

Meng Lu, Ran Xu, Yi Fang, Wenxuan Zhang, Yue Yu, Gaurav Srivastava, Yuchen Zhuang, Mohamed Elhoseiny, Charles Fleming, Carl Yang, et al. Scaling agentic reinforcement learning for tool-integrated reasoning in vlms. *arXiv preprint arXiv:2511.19773*, 2025.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. Deepswe: Training a state-of-the-art coding agent from scratch by scaling rl. https://pretty-radio-b75.notion.site/DeepSWE-Training-a-Fully-Open-sourced-State-of-the-Art-Coding-Agent-by-Scaling-RL-22281902c1468193aabbe9a8c59bbe33, 2025a. Notion Blog.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepcoder: A fully open-source 14b coder at o3-mini level. https://pretty-radio-b75.notion.site/DeepCoder-A-Fully-Open-Source-14B-Coder-at-O3-mini-Level-1cf81902c14680b3bee5eb349a512a51, 2025b. Notion Blog.

Moonshot AI. Kimi K2.7 Code. Model card, 2026. URL https://huggingface.co/moonshotai/Kimi-K2.7-Code.

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.

OpenAI. Introducing GPT-5.2, December 2025. URL https://openai.com/index/introducing-gpt-5-2/.

Sundar Pichai, Demis Hassabis, and Koray Kavukcuoglu. A new era of intelligence with Gemini 3. Google, November 2025. URL https://blog.google/products-and-platforms/products/gemini/gemini-3/.

Rushi Qiang, Yuchen Zhuang, Yinghao Li, Rongzhi Zhang, Changhao Li, Ian Shu-Hei Wong, Sherry Yang, Percy Liang, Chao Zhang, Bo Dai, et al. Mle-dojo: Interactive environments for empowering llm agents in machine learning engineering. *arXiv preprint arXiv:2505.07782*, 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.

Desik Rengarajan, Gargi Vaidya, Akshay Sarvesh, Dileep Kalathil, and Srinivas Shakkottai. Reinforcement learning with sparse rewards using guidance from offline demonstration. *arXiv preprint arXiv:2202.04628*, 2022.

Arjun Sahney, Ram Gorthi, Cezary Łastowski, and Javier Vega. Operand quant: A single-agent architecture for autonomous machine learning engineering. *arXiv preprint arXiv:2510.11694*, 2025.

Xiangru Tang, Wanghan Xu, Yujie Wang, Zijie Guo, Daniel Shao, Jiapeng Chen, Cixuan Zhang, Ziyi Wang, Lixin Zhang, Guancheng Wan, et al. Eigen-1: Adaptive multi-agent refinement with monitor-based rag for scientific reasoning. *arXiv preprint arXiv:2509.21193*, 2025.

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Maria Lupidi, et al. Ai research agents for machine learning: Search, exploration, and generalization in mle-bench. *arXiv preprint arXiv:2507.02554*, 2025.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *arXiv preprint arXiv:2502.18449*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

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*, 2025b.

Z.ai. GLM-5.3: Frontier coding with emergent cyber capabilities, August 2026a. URL https://z.ai/blog/glm-5.3.

Z.ai. GLM-5.3-Flash: Frontier intelligence, flash cost, August 2026b. URL https://z.ai/blog/glm-5.3-flash.

Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen. Acecoder: Acing coder rl via automated test-case synthesis. *arXiv preprint arXiv:2502.01718*, 2025.

Linfeng Zhang, Siheng Chen, Yuzhu Cai, Jingyi Chai, Junhan Chang, Kun Chen, Zhi X Chen, Zhaohan Ding, Yuwen Du, Yuanpeng Gao, et al. Bohrium+ scimaster: Building the infrastructure and ecosystem for agentic science at scale. *arXiv preprint arXiv:2512.20469*, 2025.

Yuxiang Zhang, Shangxi Wu, Yuqi Yang, Jiangming Shu, Jinlin Xiao, Chao Kong, and Jitao Sang. o1-coder: an o1 replication for coding. *arXiv preprint arXiv:2412.00154*, 2024.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025.

Xinyu Zhu, Yuzhu Cai, Zexi Liu, Bingyang Zheng, Cheng Wang, Rui Ye, Jiaao Chen, Hanrui Wang, Wei-Chen Wang, Yuzhi Zhang, et al. Toward ultra-long-horizon agentic science: Cognitive accumulation for machine learning engineering. *arXiv preprint arXiv:2601.10402*, 2026.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

A Implementation Details

A.1 Training settings

We initialize our policy from Qwen3.6-35B-A3B (Qwen Team, 2026a) and train it with AceGRPO to obtain Ace-35B. We use MLE-Dojo (Qiang et al., 2025) to construct our training dataset. We exclude the overlapping 68 MLE-bench tasks contained within MLE-Dojo. The final training dataset consists of 134 machine learning competition tasks. The reinforcement learning training runs for 400 steps with a rollout batch size of 8 tasks per iteration, 8 response samples per task, and a global training batch size of 64. We implement our training pipeline using the slime (Zhu et al., 2025) framework. The model was trained for two days on a cluster consisting of two nodes, each equipped with eight NVIDIA H200 GPUs. During the training phase, we utilized ten nodes, each containing eight NVIDIA RTX 4090 GPUs, to serve as execution environments for the experiments.

A.2 Overview of MLE-Bench

MLE-Bench (Chan et al., 2024) is a benchmark designed to evaluate the machine learning engineering capabilities of AI agents. The dataset consists of 75 Kaggle competitions, curated from an initial pool of over 5,000, to assess real-world skills such as model training, dataset preparation, and experiment execution. These competitions span 15 diverse domains, including natural language processing, computer vision, and signal processing. To evaluate agents across varying levels of difficulty, tasks are categorized based on the estimated time required for an experienced human engineer to solve them: 30% are Low complexity ($<$ 2 hours), 50% are Medium (2–10 hours), and 20% are High complexity ($>$ 10 hours).

MLE-Bench functions as an offline environment that replicates the rigorous conditions of actual Kaggle competitions. While original datasets are utilized whenever feasible, many competitions do not release private test sets. In such instances, new train-test splits are constructed from publicly available training data, typically reserving 10% for testing while preserving the original data distribution. Agents are required to generate a standardized `submission.csv` file for all tasks, which is evaluated using a local grading system. To unify performance assessment across diverse raw metrics, the benchmark employs the Kaggle Medal system (Bronze, Silver, Gold). Thresholds are dynamically calculated based on the original human leaderboards to align with real-world competitive standards.

MLE-Bench employs a comprehensive set of evaluation metrics. The primary indicators are the rates of achieving Any Medal, stratified by task complexity, alongside the overall average medal rate. Furthermore, the evaluation reports the proportion of tasks with valid submissions, the percentage of tasks where the model outperforms the median human competitor, and the specific acquisition rates for Bronze, Silver, and Gold medals. The specific definitions for these metrics are as follows:

- **Valid Submission:** The participant uploads a submission file that strictly adheres to the formatting requirements specified by the competition.
- **Above Median:** A valid submission achieves a leaderboard score that surpasses the performance of at least 50% of human participants.
- **Bronze Medal:** A valid submission achieves a leaderboard score that meets the official Kaggle threshold for a Bronze medal but does not reach the Silver threshold.
- **Silver Medal:** A valid submission achieves a leaderboard score that meets the official Kaggle threshold for a Silver medal but does not reach the Gold threshold.
- **Gold Medal:** A valid submission achieves a leaderboard score that meets or exceeds the official Kaggle threshold for a Gold medal.
- **Any Medal:** A valid submission achieves a leaderboard score that meets the requirements for at least a Bronze medal.

Given the prohibitive computational overhead and time constraints associated with a full evaluation on the complete MLE-Bench, the organizers of MLE-Bench have introduced MLE-Bench to facilitate more efficient benchmarking for the research community. This lightweight version comprises a curated subset of 22 Kaggle tasks selected from the original suite. Aside from the reduced task count, the evaluation protocols and metrics remain strictly consistent with the standard MLE-Bench.

B Ablation Study on Ace-30B

We conduct the ablation and training-baseline experiments using Qwen3-30B-A3B-Thinking-2507 (Yang et al., 2025a) as the base model, and denote the model trained with AceGRPO as Ace-30B. All training variants in Table 2, including SFT, Vanilla GRPO, and the AceGRPO component ablations, start from this same base model. This shared initialization allows us to compare training methods and assess the contributions of the Evolving Data Buffer and Adaptive Sampling. These experiments complement the main results for Ace-35B in Table 1, which uses Qwen3.6-35B-A3B as its base model.

Table 2: Ablation and training-baseline study for Ace-30B on MLE-Bench-Lite. Component ablations use a 400-step RL budget. Δ: Any Medal gain over Qwen3-30B.

Method	Any Medal	HumanRank	Δ
Qwen3-30B-A3B-2507	27.27%	48.75	-
SFT	36.36%	57.80	+9.09%
Vanilla GRPO	34.85%	55.90	+7.58%
AceGRPO	**51.52%**	**71.14**	**+24.25%**
w/o Evolving Data Buffer	45.45%	65.80	+18.18%
w/o Adaptive Sampling	42.73%	62.70	+15.46%

We compare AceGRPO with an SFT baseline, Vanilla GRPO, and two component ablations that remove the Evolving Data Buffer or Adaptive Sampling. Additional hyperparameter sensitivity analysis is reported in Section C.4. We highlight two main conclusions below.

AceGRPO is more effective than SFT from successful trajectories. We train an SFT baseline. The SFT data comes from the AceGRPO training rollout trajectories, and contains 14K samples distilled from DRAFT, DEBUG, and IMPROVE states. Table 2 shows SFT improves Any Medal from 27.27% to 36.36%, showing that successful trajectories provide useful supervision. However, AceGRPO reaches 51.52% Any Medal and 71.14 HumanRank, exceeding SFT by 15.16 percentage points in Any Medal and 13.34 HumanRank points. This demonstrates that AceGRPO's online execution-grounded training provides a stronger signal than static imitation, especially because the policy learns from its own intermediate states, failures, and recovery attempts.

The evolving data buffer and adaptive sampling are both essential to AceGRPO. All RL variants are trained with a matched 400-step budget. As shown in Table 2, both components are critical to AceGRPO's performance. Removing the **Evolving Data Buffer** reduces the Any Medal rate by 6.07 percentage points, highlighting the importance of streaming task expansion. Without dynamically repurposing intermediate execution traces into new training tasks, the agent is confined to repetitive trajectories and cannot self-evolve beyond the static dataset distribution in the absence of on-policy interaction. Removing **Adaptive Sampling** causes an even larger 8.79-percentage-point drop in Any Medal rate. This underscores the value of Adaptive Sampling and our proposed Learnability Potential Function: as the task pool grows, naive sampling becomes increasingly inefficient because the learning signal is diluted by already-mastered or effectively intractable tasks. **Vanilla GRPO**, which lacks both mechanisms, obtains a 7.58-percentage-point improvement over the base LLM. These results show that AceGRPO's improvement is not simply due to applying GRPO, but comes from coupling online state evolution with adaptive sampling of informative training states.

C Hyperparameters

This section summarizes the main hyperparameters used for AceGRPO training. We report the algorithmic and optimization settings that materially affect the method, and omit infrastructure-only details such as sandbox retry limits, polling intervals, and logging options.

Table 3: Key hyperparameters for buffer construction and adaptive sampling.

Parameter	Symbol	Value	Description
Initial potential	$\mathcal{P}_{\text{init}}$	0.1	Default potential for newly inserted states before their first rollout
Std. deviation clip	$\delta_{\max}$	1.0	Upper bound for the uncertainty term in the learnability potential
Initial temperature	τ_0	1.0	Initial temperature for potential-based sampling
Temperature floor	ϵ_τ	0.1	Minimum temperature ratio in the annealing schedule
Saturation buffer size	N_{sat}	128	Buffer size at which the sampler reaches the temperature floor
Recency window	Δ	10	Number of rollout steps over which recently sampled states are down-weighted
Max buffer size	$N_{\max}$	128	Maximum number of state groups maintained in the evolving buffer
Improvement reward weight	α	0.3	Weight of the relative-improvement term in Equation (6)

Table 4: GRPO rollout configuration.

Parameter	Value	Description
Group size	$G = 8$	Number of responses sampled from each starting state
Rollout batch size	$B = 8$	Number of starting states sampled per rollout step
Global batch size	64	Number of generated samples used per policy update ($B \times G$)
Rollout temperature	0.7	Sampling temperature for policy generation
Maximum response length	16,384	Maximum number of generated tokens per response
Main training steps	400	Number of rollout/update iterations for the main Ace-35B run

C.1 Buffer and Adaptive Sampling

Table 3 lists the hyperparameters for the evolving data buffer and the adaptive sampling distribution in Sections 4.1 and 4.2. AceGRPO uses a smooth temperature schedule and a recency factor rather than a hand-designed multi-stage curriculum.

C.2 GRPO Rollout Configuration

Table 4 lists the rollout-level settings used to construct each GRPO training batch.

C.3 Policy Optimization

Table 5 provides the policy optimization settings.

Table 5: Policy optimization hyperparameters.

Parameter	Value	Description
Optimizer	Adam	Adaptive moment estimation optimizer
Learning rate	1×10^{-6}	Constant learning rate for policy updates
Weight decay	0.1	L2 regularization coefficient
Adam β_1	0.9	Exponential decay rate for the first moment estimate
Adam β_2	0.98	Exponential decay rate for the second moment estimate
PPO clip range	$[0.2, 0.28]$	Lower and upper clipping bounds used in the GRPO/PPO-style objective
KL loss coefficient	0.005	Coefficient for the KL penalty against the reference policy
Entropy coefficient	0.0005	Coefficient for entropy regularization

C.4 Hyperparameter Ablation on Ace-30B

For the earlier Ace-30B model, we conduct hyperparameter ablation on the reward mixing coefficient α in Equation (6). This is because α is the only hyperparameter that directly changes the learning

Table 6: Hyperparameter ablation for the earlier Ace-30B model, varying the reward mixing coefficient α on MLE-Bench. Δ denotes the Any Medal change relative to the base Qwen3-30B model. Dashes indicate metrics not included in this sensitivity summary.

Model / Setting	Steps	Valid	Above Med.	HumanRank	Any Medal	Δ
Qwen3-30B baseline	–	84.85	50.00	48.75	27.27	–
$\alpha = 0.0$	200	92.42	46.97	46.52	19.70	-7.57%
$\alpha = 0.6$	200	93.18	50.00	51.69	29.55	+2.28%
$\alpha = 0.3$ (AceGRPO)	**200**	**96.97**	**56.06**	**59.37**	**37.30**	**+10.03%**

objective: it controls the trade-off between optimizing absolute solution quality and encouraging relative improvement over the current state. The sensitivity runs use the same Qwen3-30B-A3B-Thinking-2507 initialization, the same MLE-Dojo training set, the same two-node training layout, and the same AceGRPO configuration, changing only `reward.alpha_improve`. For efficiency, these runs are trained for 200 steps.

In contrast, the remaining hyperparameters in Tables 3 to 5 are not separately ablated. Parameters such as N_{sat}, Δ, and $N_{\max}$ specify the operating scale of the buffer and sampler: N_{sat} determines when temperature annealing reaches its floor, Δ controls short-term recency suppression, and $N_{\max}$ bounds the training-side state pool. Similarly, the group size, rollout batch size, and maximum response length determine the execution budget per update, while optimizer settings follow stable GRPO/PPO-style fine-tuning defaults. Ablating these quantities would mostly mix algorithmic effects with compute budget, wall-clock throughput, memory pressure, or buffer-capacity constraints, rather than testing a distinct methodological claim. We therefore use the component ablations in Appendix B to evaluate the data buffer and adaptive sampler, and reserve hyperparameter sensitivity for α.

Table 6 summarizes the 200-step sensitivity study. With $\alpha = 0$, the reward reduces to the absolute HumanRank score and the Any Medal rate drops below the base model, suggesting that absolute quality alone provides insufficient pressure for iterative refinement. Increasing the weight to $\alpha = 0.6$ recovers performance and slightly improves over the base model, but still underperforms the default $\alpha = 0.3$ setting at the same 200-step checkpoint. This indicates that AceGRPO benefits from relative-improvement feedback, but overly emphasizing improvement over the parent state can make the reward more sensitive to local baselines and noisy small gains. We therefore use $\alpha = 0.3$ as the default setting, which provides the best trade-off in our runs.

D LICENSING

All codebases, models and data used in this work are publicly available. We have ensured compliance with the corresponding licenses of these repositories, respecting all terms of use and attribution requirements.

E PROMPTS

Prompts for drafting an initial code

```
You are a Kaggle grandmaster attending a competition. In order to win this competition,
you need to come up with an excellent and creative plan for a solution and then
implement this solution in Python. We will now provide a description of the task.

# Task description
{task_description}

# Instructions
## Response format
Your response should be a brief outline/sketch of your proposed solution in natural
language (3-5 sentences), followed by a single markdown code block (wrapped in “‘)
which implements this solution and prints out the evaluation metric. There should be
no additional headings or text in your response. Just natural language text followed
by a newline and then the markdown code block.

## Solution sketch guideline
- The solution sketch should be 3-5 sentences.
- Propose an evaluation metric that is reasonable for this task.
- Don’t suggest to do EDA.
- The data is already prepared and available in the ‘./input‘ directory. There is no
need to unzip any files.

## Implementation guideline
- The code must not only implement the proposed solution but also **print the evaluation
metric computed on a hold-out validation set**. **Without this metric, the solution
cannot be evaluated, rendering the entire code invalid.**,
- **AND MOST IMPORTANTLY SAVE PREDICTIONS ON THE PROVIDED UNLABELED TEST DATA IN
A ‘submission.csv‘ FILE IN THE ./submission/ DIRECTORY.** - The code should be a
single-file python program that is self-contained and can be executed as-is.
- No parts of the code should be skipped, don’t terminate the before finishing the
script.
- Your response should only contain a single code block.
- All the provided input data is stored in "./input" directory.
- **You MUST submit predictions on the provided unlabeled test data in a ‘submission.csv‘
file** file in the "./working" directory as described in the task description** This
is extremely important since this file is used for grading/evaluation. DO NOT FORGET
THE submission.csv file!
- You can also use the "./working" directory to store any temporary files that your
code needs to create.
- REMEMBER THE ./submission/submission.csv FILE!!!!! The correct directory is important
too.
- If you use ‘DataLoader‘, you need to increase the parameter ‘num_workers‘ to speed
up the training process.

## Installed Packages
Your solution can use any relevant machine learning packages such as: ‘pandas‘,
‘statsmodels‘, ‘torch-geometric‘, ‘bayesian-optimization‘, ‘torch‘, ‘xgboost‘,
‘spacy‘, ‘timm‘, ‘scikit-learn‘, ‘transformers‘, ‘nltk‘, ‘lightGBM‘, ‘numpy‘,
‘torchvision‘. Feel free to use any other packages too (all packages are already
installed!). For neural networks we suggest using PyTorch rather than TensorFlow.

# Data preview
```

```
{data_preview}
```

Prompts for debugging

```
You are a Kaggle grandmaster attending a competition. Your previous solution had a
bug and/or did not produce a submission.csv, or the generated submission.csv was in
an incorrect format,so based on the information below, you should revise it in order
to fix this. Your response should be an implementation outline in natural language,
followed by a single markdown code block which implements the bugfix/solution.

# Task description
{task_description}

# Instructions
## Response format
Your response should be a brief outline/sketch of your proposed solution in natural
language (3-5 sentences), followed by a single markdown code block (wrapped in “‘)
which implements this solution and prints out the evaluation metric. There should be
no additional headings or text in your response. Just natural language text followed
by a newline and then the markdown code block.

## Bugfix improvement sketch guideline
- - You should write a brief natural language description (3-5 sentences) of how the
issue in the previous implementation can be fixed.
- - Don’t suggest to do EDA.
- - You should keep the core method of the machine learning code same. Do not change
the machine learning method and just fix the code.
- - If the code failed because of missing library, try to avoid using the missing
library. Do not try to install the missing library.
- - All packages have been installed. You are not allowed to install anything with
pip or conda. If something is missing, try another way instead of installing a package.

## Implementation guideline
- The code must not only implement the proposed solution but also **print the evaluation
metric computed on a hold-out validation set**. **Without this metric, the solution
cannot be evaluated, rendering the entire code invalid.**,
- **AND MOST IMPORTANTLY SAVE PREDICTIONS ON THE PROVIDED UNLABELED TEST DATA IN A
‘submission.csv‘ FILE IN THE ./submission/ DIRECTORY.**
- The code should be a single-file python program that is self-contained and can be
executed as-is.
- No parts of the code should be skipped, don’t terminate the before finishing the
script.
- Your response should only contain a single code block.
- All the provided input data is stored in "./input" directory.
- **You MUST submit predictions on the provided unlabeled test data in a ‘submission.csv‘
file** file in the "./working" directory as described in the task description** This
is extremely important since this file is used for grading/evaluation. DO NOT FORGET
THE submission.csv file!
- You can also use the "./working" directory to store any temporary files that your
code needs to create.
- REMEMBER THE ./submission/submission.csv FILE!!!!! The correct directory is important
too.
- If you use ‘DataLoader‘, you need to increase the parameter ‘num_workers‘ to speed
up the training process.

# Data preview
{data_preview}

# Previous (buggy) implementation
{buggy_code}

# Execution output
```

```
{terminal_output}
```

Prompts for improving

```
You are a Kaggle grandmaster attending a competition. You are provided with previous
memory including previously developed solutions and an creative idea. You need to
implement this idea on top of (or building upon) the previously developed solution
and memory.

# Task description
Here is the original kaggle task description.
{task_description}

# Instructions
Here is the instruction about response format and implementation.
## Response format
Your response should be a brief outline/sketch of the solution in natural language
(3-5 sentences), followed by a single markdown code block (wrapped in “‘) which
implements this solution and prints out the evaluation metric. There should be no
additional headings or text in your response. Just natural language text followed by
a newline and then the markdown code block.

## Solution improvement sketch guideline
- - The solution sketch should be a brief natural language description of how you
improved the previous solution.
- - The solution sketch should be 3-5 sentences.
- - Don’t do EDA.
- - All packages have been installed. You are not allowed to install anything with
pip or conda. If something is missing, try another way instead of installing a package.

## Implementation guideline
- The code must not only implement the creative idea but also **print the evaluation
metric computed on a hold-out validation set**. **Without this metric, the solution
cannot be evaluated, rendering the entire code invalid.**,
- **AND MOST IMPORTANTLY SAVE PREDICTIONS ON THE PROVIDED UNLABELED TEST DATA IN A
‘submission.csv‘ FILE IN THE ./submission/ DIRECTORY.**
- The code should be a single-file python program that is self-contained and can be
executed as-is.
- No parts of the code should be skipped, don’t terminate the before finishing the
script.
- Your response should only contain a single code block.
- All the provided input data is stored in "./input" directory.
- **You MUST submit predictions on the provided unlabeled test data in a ‘submission.csv‘
file** file in the "./working" directory as described in the task description** This
is extremely important since this file is used for grading/evaluation. DO NOT FORGET
THE submission.csv file!
- You can also use the "./working" directory to store any temporary files that your
code needs to create.
- REMEMBER THE ./submission/submission.csv FILE!!!!! The correct directory is important
too.
- If you use ‘DataLoader‘, you need to increase the parameter ‘num_workers‘ to speed
up the training process.

# Data preview {data_preview}
```

F Additional Evaluation over Time

Figure 6 reports all seven models at the supplied time checkpoints under the same 12-hour MLE-Bench-Lite evaluation protocol. The main-text Figure 3 selects the trained model and its base, DeepSeek-V4-Pro as a close frontier comparator, and GLM-5.3 as the highest final medal-rate model in this comparison. This larger figure includes the remaining models for completeness. The curves show reported means, with straight segments connecting the observed checkpoints and no smoothing.

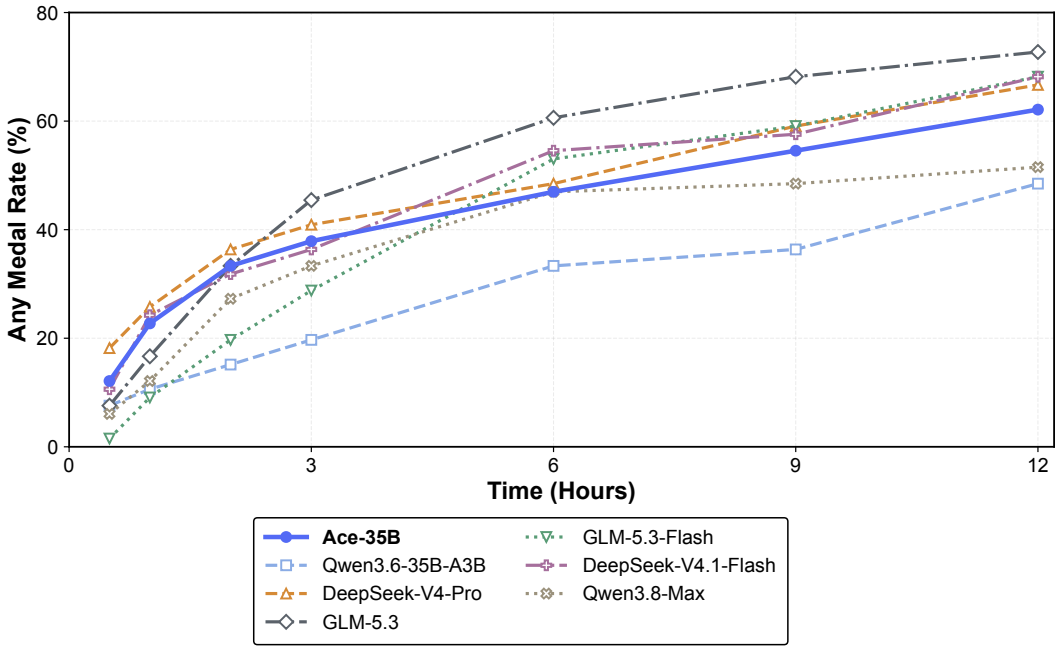

Figure 6: Any Medal rate over time for all seven models on MLE-Bench-Lite. Markers denote the reported means at 0.5, 1, 2, 3, 6, 9, and 12 hours.