AceGRPO: Adaptive Curriculum Enhanced Group Relative Policy Optimization for Autonomous Machine Learning Engineering

Anonymous ACL submission

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. Leveraging AceGRPO, our trained **Ace-30B** model achieves a **100% valid submission rate** on MLE-Bench-Lite, approaches the performance of proprietary frontier models, and outperforms larger open-source baselines (e.g., DeepSeek-V3.2), demonstrating robust capability for sustained iterative optimization.

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., 2025b). 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 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., 2025c; 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.

Empirically, AceGRPO improves both sample efficiency and overall performance. On OpenAI's MLE-Bench-Lite (Chan et al., 2024), Ace-30B (trained with AceGRPO) establishes a new state-of-the-art among open-source large models across all metrics, while also exceeding the performance of proprietary frontier models on specific indicators. Specifically, we observe a 12.13% improvement in medal rate compared to Deepseek V3.2 and a 24.25% gain relative to the untrained baseline. Extensive analysis and ablation experiments confirm the robustness of AceGRPO and underscore the critical role of the Evolving Data Buffer and Adaptive Sampling. And 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**, our Ace-30B achieves a **100% valid submission rate** and shows stronger iterative optimization behavior than substantially larger frontier baselines.

2 Related Work

2.1 Autonomous Machine Learning Engineering

LLM-driven agents have advanced Machine Learning Engineering (MLE) from static QA toward iterative, execution-guided optimization. Benchmarks and platforms such as MLE-Bench (Chan et al., 2024) and OpenHands (Wang et al., 2024) highlight the need for tight feedback loops, while recent MLE agents rely largely on inference-time exploration, including prompt-based planning (Jiang et al., 2025; Yang et al., 2025b; Liu et al., 2025b), evolutionary search (Toledo et al., 2025; Du et al., 2025a; Li et al., 2025), tool optimization (Sahney et al., 2025), targeted initialization (Nam et al., 2025), and context management (Zhu et al., 2026). However, these parameter-frozen methods cannot internalize successful strategies, leading to limited self-improvement. AceGRPO addresses this limitation by training agents over a dynamic task pool that prioritizes learnable intermediate states, enabling sample-efficient self-evolution.

2.2 Agentic Reinforcement Learning for LLMs

Agentic Reinforcement Learning treats LLMs as learnable policies in sequential decision loops rather than static text generators (Cao et al., 2024). It has enabled slow reasoning and self-correction in models such as DeepSeek-R1 (Guo et al., 2025) and o1-Coder (Zhang et al., 2024), and has been

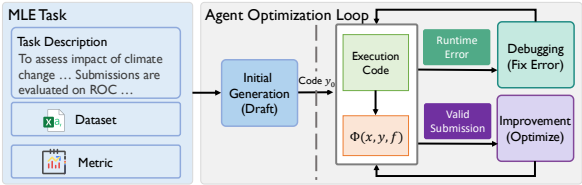

Figure 1: The Agent Optimization Loop for MLE. It is a continuous iteration over three distinct phases: Draft (initial generation), Debug (error correction), and Improve (metric optimization). The transitions are deterministically governed by the execution feedback, allowing the agent to refine its code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

extended to software engineering through systems including AceCoder (Zeng et al., 2025b), DeepCoder (Luo et al., 2025b), DeepSWE (Luo et al., 2025a), SWE-RL (Wei et al., 2025), and Qwen3-Coder (Yang et al., 2025a). In MLE, ML-Agent (Liu et al., 2025c) demonstrates the potential of RL for automated machine learning pipelines. Yet long-horizon MLE remains bottlenecked by costly full-trajectory rollouts and the limited exploration of static datasets (Zhao et al., 2025; Golubev et al., 2025). AceGRPO mitigates this bottleneck with an adaptive curriculum that focuses training on high-potential intermediate states.

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:

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

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 $\mathcal{B}_t$:

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

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

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

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

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

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:

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

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 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., $\text{supp}(\mathcal{Q}_t) \subseteq \mathcal{B}_t$.

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* $\kappa(x) \in \{\text{DRAFT}, \text{DEBUG}, \text{IMPROVE}\}$, 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.

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

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

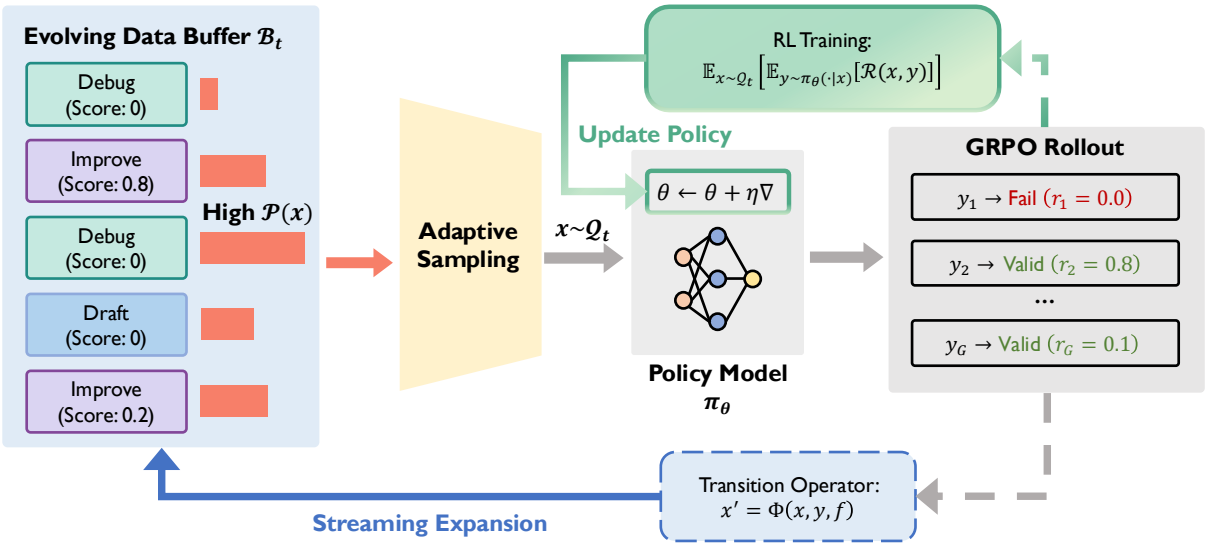

Figure 2: The AceGRPO framework. MLE is step-wise optimization over a dynamic task distribution. The method features an Evolving Data Buffer that accumulates intermediate states (Draft, Debug, Improve) via Streaming Expansion. An Adaptive Sampler selects high-potential tasks to maximize the gradient signal for GRPO training, enabling efficient self-improvement on long-horizon challenges.

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:

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

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,

Table 1: Main evaluation results on MLE-Bench-Lite. All results are reported as mean ± standard deviation. The best open-source model performance is marked **bold**. Qwen3-235B-2507 refers to Qwen3-235B-A22B-Thinking-2507, and Qwen3-30B-2507 refers to Qwen3-30B-A3B-Thinking-2507. Except for the HumanRank Score multiplied by 100 for readability (defined in $[0, 1]$), all other metrics are reported as percentages.

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-4.6	90.91 ± 3.71	45.45 ± 3.71	6.06 ± 4.29	3.03 ± 2.14	21.21 ± 4.29	30.30 ± 3.71	45.45 ± 7.73
DeepSeek-V3.2	98.48 ± 2.14	71.21 ± 2.14	7.58 ± 2.14	4.55 ± 3.71	27.27 ± 3.71	39.39 ± 2.14	65.92 ± 1.62
Qwen3-235B-2507	92.42 ± 2.14	69.70 ± 7.73	6.06 ± 5.67	7.58 ± 4.29	24.24 ± 4.29	37.88 ± 10.71	69.70 ± 7.73
Qwen3-30B-2507	84.85 ± 2.14	50.00 ± 3.71	1.52 ± 2.14	7.58 ± 2.14	18.18 ± 3.71	27.27 ± 6.43	48.75 ± 3.45
Ace-30B	**100.00 ± 0.00**	**78.79 ± 5.67**	**12.12 ± 4.29**	**7.58 ± 2.14**	**31.82 ± 3.71**	**51.52 ± 8.57**	**71.14 ± 4.08**

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.

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. In our experiments, we fine-tuned Qwen3-30B-A3B-Thinking-2507 (Yang et al., 2025a) using our AceGRPO to obtain Ace-30B. 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). 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 rollouts per task, and a global training batch size of 64. Additional training details are provided in Appendix C.1. And prompts are in Appendix G

Evaluation settings. We evaluate our trained models on MLE-Bench-Lite, a 22-task Kaggle-style subset of MLE-Bench (Chan et al., 2024). Each competition presents a realistic Kaggle-style challenge where the agent must analyze the problem, train models, and submit predictions—all within a 12-hour time budget per task. Following MLE-Bench, we measure performance using the following metrics: *valid submission*, *above-median*, *bronze*, *silver*, *gold*, and *any medal*. Additionally, we report the *HumanRank score* from MLE-Dojo (Qiang et al., 2025). Details about

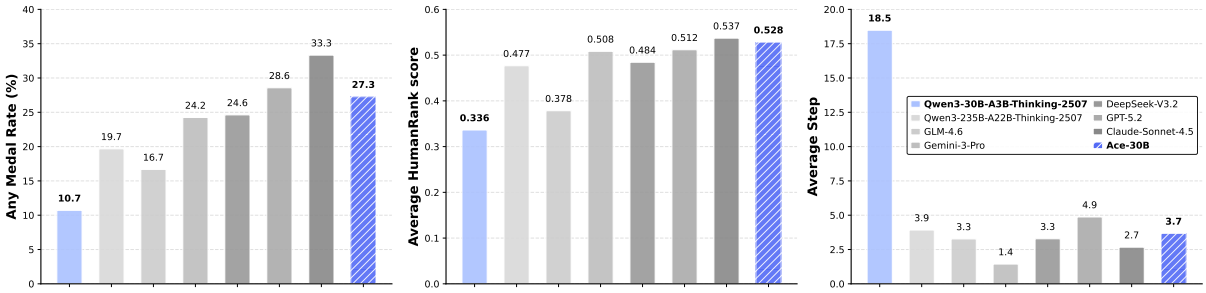

Figure 3: Performance comparison on first valid submissions. We report: (left) medal rate on the first valid submission, (middle) average HumanRank score of the first valid submission, and (right) average number of agent steps needed to produce the first valid submission. Lower steps indicate faster generation of valid solutions.

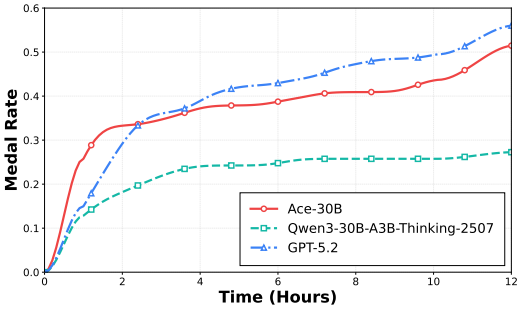

Figure 4: Medal rate evolution over time.

metrics and MLE-Bench are provided in Appendix C.2. We compare our approach against several strong baselines: (1) proprietary frontier models including Claude-4.5-Sonnet (Anthropic, 2025), Gemini-3-Pro (Google, 2025), and GPT-5.2 (OpenAI, 2025); (2) open-source models including DeepSeek-V3.2 (Liu et al., 2025a), GLM 4.6 (Zeng et al., 2025a), and Qwen3-235B-A22B-Thinking-2507 (Yang et al., 2025a); and (3) the base Qwen3-30B-A3B-Thinking-2507 without RL training. We enable the thinking mode of all the baseline models and our Ace-30B in the evaluation. To ensure statistical reliability, we repeat each evaluation run three times and report mean performance with standard deviation. Ablation studies are reported in Appendix D.

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-30B consistently outperforms the baseline Qwen3-30B-A3B-Thinking model in all major evaluation metrics. In particular, the Any Medal rate increases by 24.25% (from 27.27% to 51.52%), and the HumanRank score improves by 22.39% (from 48.75 to 71.14). Notably, Ace-30B also achieves a **100% Valid Submission rate**, matching the strongest proprietary LLM (Claude-4.5-Sonnet) 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.

AceGRPO approaches the performance of proprietary frontier LLMs while outperforming larger open-source baselines. As shown in Table 1, Ace-30B attains aggregated performance that is competitive with proprietary frontier models on both Any Medal rate and HumanRank score, despite being trained from a substantially smaller open-source LLM. In particular, Ace-30B reaches a 51.52% Any Medal rate and a 71.14% HumanRank score, matching the level of GPT-5.2 (56.06% Any Medal; 71.05 HumanRank score) and even slightly surpassing it on Human Rank. Meanwhile, Ace-30B clearly outperforms larger open-source baselines such as DeepSeek-V3.2 (39.39% Any Medal; 65.92 HumanRank score) and Qwen3-235B-2507 (37.88% Any Medal; 69.70 HumanRank score). These results suggest that AceGRPO closes a substantial portion of the gap to closed-source frontier LLMs and delivers stronger performance than much larger open-source models under realistic MLE evaluation settings.

AceGRPO enables sustained self-evolution rather than one-shot performance gains. Figure 4 plots the Any Medal rate over time on MLE-Bench-Lite for different LLMs. Whereas Qwen3-30B-A3B-Thinking largely plateaus within the first six hours, Ace-30B continues to improve steadily throughout the entire run, consistently surpassing the base model and progressively narrowing the gap to GPT-5.2. This persistent upward trajectory suggests that AceGRPO facilitates sustained self-

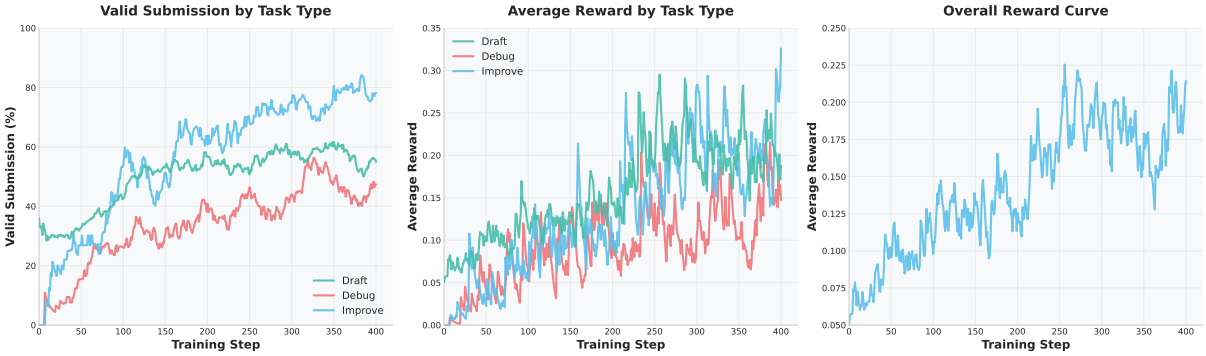

Figure 5: Training dynamics of rewards curve and performance of valid submission.

improvement over extended horizons, rather than yielding isolated or brittle gains, supporting its effectiveness for MLE-style tasks.

5.3 Analysis

AceGRPO significantly improves both the quality and efficiency of early-stage solutions. Figure 3 compares different LLMs based on their first valid submission, and reports three complementary metrics: Any Medal rate, average HumanRank score, and the number of steps required to reach the first valid solution. These metrics jointly characterize not only the quality of the initial feasible solutions but also the efficiency with which an LLM escapes invalid states. Ace-30B achieves a substantially higher medal rate on the first valid submission, improving from 10.71% for the base Qwen3-30B-A3B-Thinking model to 27.27%. At the same time, it significantly reduces the average number of steps required to reach a valid submission from 18.48 down to 3.67. This indicates that AceGRPO enables LLMs to generate valid and competitive solutions much earlier in the optimization process, rather than relying on prolonged trial-and-error to recover from failures. Notably, Ace-30B's first-submission performance is competitive with larger frontier LLMs such as Claude-4.5-Sonnet and GPT-5.2, both in terms of solution quality and solution emergence speed. This suggests that the improvements introduced by AceGRPO are not limited to final performance but also manifest strongly at early stages of interaction, where efficient exploration and rapid stabilization are critical (Zhu et al., 2026). Overall, Figure 3 demonstrates that AceGRPO enhances early-stage robustness and efficiency, enabling LLM-based MLE agents to transition more reliably and quickly from invalid states to meaningful optimization trajectories.

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 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.

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 a dynamically evolving task distribution. By prioritizing intermediate states in the agent's "Learning Zone" via *Learnability Potential*, AceGRPO allocates limited computation to states with informative gradient signals and reduces redundant exploration.

On MLE-Bench-Lite, Ace-30B trained with AceGRPO achieves 100% Valid Submission, a 51.52% medal rate, and a 0.7114 HumanRank score, outperforming all evaluated open-source LLMs. Notably, it enables a 30B model to exhibit optimization behavior and final performance comparable to proprietary frontier models such as GPT-5.2. These results suggest a promising path toward efficiently training self-evolving agents for long-horizon MLE tasks.

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.

References

Anthropic. 2025. Introducing claude sonnet 4.5. System card, Anthropic.

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

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, and 1 others. 2024. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*.

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

Yihe Deng, I Hsu, Jun Yan, Zifeng Wang, Rujun Han, Gufeng Zhang, Yanfei Chen, Wei Wang, Tomas Pfister, Chen-Yu Lee, and 1 others. 2025. Supervised reinforcement learning: From expert trajectories to step-wise reasoning. *arXiv preprint arXiv:2510.25992*.

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

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

Alexander Golubev, Maria Trofimova, Sergei Polezhaev, Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Sergey Abramov, Andrei Andriushchenko, Filipp Fisin, and 1 others. 2025. Training long-context, multi-turn software engineering agents with reinforcement learning. *arXiv preprint arXiv:2508.03501*.

Google. 2025. Gemini 3 pro best for complex tasks and bringing creative concepts to life. System card, Google.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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

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

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

Annan Li, Chufan Wu, Zengle Ge, Yee Hin Chong, Zhinan Hou, Lizhe Cao, Cheng Ju, Jianmin Wu, Huaiming Li, Haobo Zhang, and 1 others. 2025. The fm agent. *arXiv preprint arXiv:2510.26144*.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025a. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Siheng Chen, and 1 others. 2025b. Ml-master: Towards ai-for-ai via integration of exploration and reasoning.

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

Meng Lu, Ran Xu, Yi Fang, Wenxuan Zhang, Yue Yu, Gaurav Srivastava, Yuchen Zhuang, Mohamed Elhoseiny, Charles Fleming, Carl Yang, and 1 others. 2025. Scaling agentic reinforcement learning for tool-integrated reasoning in vlms. *arXiv preprint arXiv:2511.19773*.

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. 2025a. 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. 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. 2025b. 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. Notion Blog.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Ö Arık, and Tomas Pfister. 2025. Mle-star: Machine learning engineering agent via search and targeted refinement. *arXiv preprint arXiv:2506.15692*.

OpenAI. 2025. Gpt-5 system card. System card, OpenAI.

Rushi Qiang, Yuchen Zhuang, Yinghao Li, Rongzhi Zhang, Changhao Li, Ian Shu-Hei Wong, Sherry Yang, Percy Liang, Chao Zhang, Bo Dai, and 1 others. 2025. Mle-dojo: Interactive environments for empowering llm agents in machine learning engineering. *arXiv preprint arXiv:2505.07782*.

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

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

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

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Maria Lupidi, and 1 others. 2025. Ai research agents for machine learning: Search, exploration, and generalization in mle-bench. *arXiv preprint arXiv:2507.02554*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, and 1 others. 2024. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Xu Yang, Xiao Yang, Shikai Fang, Bowen Xian, Yuante Li, Jian Wang, Minrui Xu, Haoran Pan, Xinpeng Hong, Weiqing Liu, and 1 others. 2025b. R&d-agent: Automating data-driven ai solution building through llm-powered automated research, development, and evolution. *arXiv preprint arXiv:2505.14738*.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025a. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

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

Linfeng Zhang, Siheng Chen, Yuzhu Cai, Jingyi Chai, Junhan Chang, Kun Chen, Zhi X Chen, Zhaohan Ding, Yuwen Du, Yuanpeng Gao, and 1 others. 2025. Bohrium+ scimaster: Building the infrastructure and ecosystem for agentic science at scale. *arXiv preprint arXiv:2512.20469*.

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

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

Xinyu Zhu, Yuzhu Cai, Zexi Liu, Bingyang Zheng, Cheng Wang, Rui Ye, Jiaao Chen, Hanrui Wang, Wei-Chen Wang, Yuzhi Zhang, and 1 others. 2026. Toward ultra-long-horizon agentic science: Cognitive accumulation for machine learning engineering. *arXiv preprint arXiv:2601.10402*.

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

A LLM Usage Disclosure

We used LLM solely to improve the readability and fluency of the manuscript. All research ideas, experimental design, implementation, analysis, and conclusions are our own. The authors have reviewed and verified the final manuscript and take full responsibility for its content.

B Related Work

B.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., 2025b) 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 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.

B.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., 2025b) 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., 2025c) 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.

C Implementation Details

C.1 Training settings

In our experiments, we train the Qwen3-30B-A3B-Thinking-2507 model using our AceGRPO. 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.

C.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.

D Ablation Study

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 Appendix E.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, substantially outperforming SFT by 15.16% in Any Medal and 13.34% in

Table 2: Ablation and training-baseline study on MLE-Bench-Lite. Component ablations use the same 400-step RL budget. Δ denotes the improvement in Any Medal rate over the base Qwen3-30B model.

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%

HumanRank. 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%, 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% 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 only a 7.58% 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.

E 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.

E.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.

E.2 GRPO Rollout Configuration

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

E.3 Policy Optimization

Table 5 provides the policy optimization settings.

E.4 Hyperparameter Ablation

We only 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 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.

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	520	Number of rollout/update iterations for the main AceGRPO run

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 component ablations in the main text 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.

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

Table 6: Hyperparameter ablation for 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%**

F 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.

G 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}
```