VERIFY LESS, EVOLVE MORE: TRAINING IDEA-LEVEL CRITICS FOR VERIFICATION-EFFICIENT ML EVOLVING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

As large language models become more powerful, self-evolving agents are able to tackle challenging tasks including AI for machine learning (AI4ML). In AI4ML, while empirical verification is available, it often requires computationally costly model training and evaluation, limiting the speed and scale of agent evolution. Yet verification efficiency remains underexplored, and frontier models provide only limited gains when used directly as idea selectors. We address this gap with specialized *idea-level critic models* that predict whether a proposed ML modification will improve upon the current solution, allowing agents to screen ideas and concentrate verification resources on the most promising candidates. We train the critic models through supervised fine-tuning on high-quality critiques synthesized by Gemini-3.1-Pro, followed by GRPO to further improve their predictive accuracy. Empirically, our critic models outperform Gemini-3.1-Pro in static idea evaluation, and these gains extend to agent inference, continual learning, and policy training. During inference-time evolution, they improve final solution quality under the same verification budget by selecting more promising ideas, with further gains from continual learning. During policy training, they serve as learned reward models, reserving empirical verification for uncertain cases and enabling substantially more policy updates with the same verification resources. Together, these results show that idea-level critic models help ML agents discover better solutions and learn stronger proposal policies under limited verification budgets.

1 INTRODUCTION

Recent advances in large language models (LLMs) have spurred growing interest in self-improving and evolutionary agents, which iteratively propose modifications and retain successful variants based on empirical feedback (Novikov et al., 2025; Zhang et al., 2026a; Sharma, 2025). These efforts demonstrate unprecedented success on challenging tasks such as AI for machine learning (AI4ML), where agents develop and refine ML solutions through implementation and empirical evaluation (Huang et al., 2023; Chan et al., 2025; Lu et al., 2024). However, AI4ML presents a distinctive verification challenge: while available, empirical verifiers often require computationally costly model training and evaluation. This execution cost accumulates across candidates and evolutionary rounds, directly limiting how many alternatives an agent can explore under a fixed compute budget. Addressing this issue by simply using a frontier model to screen candidates offers limited relief: in our OpenEvolve experiments, directly prompting Gemini-3.1-Pro as an idea selector yields only modest gains (Section 4). These limitations motivate us to ask: can we train specialized models to assess proposed ML modifications before empirical verification, reserving expensive execution for those most likely to improve the current solution?

In this work, we develop a training pipeline for specialized *idea-level critic models* that screens proposed ML improvements before verification, reducing reliance on computationally costly empirical verification. Given a task description, a parent solution, and a proposed child modification, the critic model predicts whether the modification will improve upon the parent. To train the critic model, we collect realistic idea-evolution trajectories from live ML agent runs and label parent–child pairs using their empirical execution outcomes. We then use Gemini-3.1-Pro (Google Gemini Team, 2026) to synthesize critiques, retain critiques consistent with the execution-derived labels through rejection

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

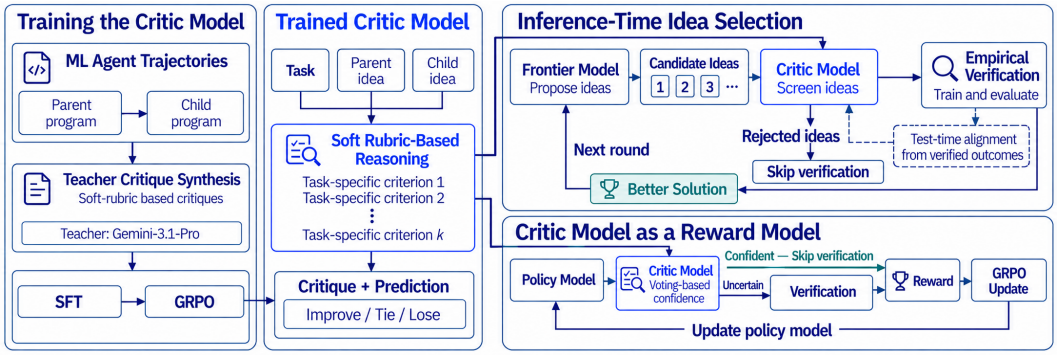


Figure 1: We train an idea-level critic model from ML agent evolution trajectories through SFT on soft-rubric critiques followed by GRPO. The trained critic model then supports (1) inference-time evolution through idea selection and task-specific test-time alignment, and (2) policy training by serving as a learned reward model while routing uncertain cases to empirical verification.

sampling, and train the critic model through supervised fine-tuning followed by GRPO (Shao et al., 2024). The critiques follow a soft rubric-based reasoning format: the critic model compares the parent and child under every task-specific criterion while retaining flexibility in weighting the criteria and combining their judgments. Our ablations show that this structure supports continued improvement during further RL alignment, whereas unconstrained reasoning leads to performance degradation. We hold out a portion of the collected data to form ML-IDEA-CRITIC-BENCH, a static benchmark for measuring training gains and comparing the idea-level assessment capabilities of different models. On this benchmark, our Qwen-3.5-9B and Qwen-3.5-35B critic models achieve three-way directional accuracies of 60.7% and 63.3%, respectively, substantially outperforming Gemini-3.1-Pro at 46.6%.

We evaluate how these predictive gains translate into improvements in agent inference, continual learning, and policy training. First, integrated into OpenEvolve (Sharma, 2025) with Gemini-3.1-Pro as the idea proposer, the critic models screen candidate ideas before verification, improving final solution quality on MLS-Lite-Bench under matched verification budgets. Second, continual learning allows the critic models to adapt using empirical feedback accumulated during the current evolution run, further improving the final solutions. Third, while critic model improves evolution by prioritizing the most promising candidates, the quality of ideas generated remain constraint by the fixed proposer. We therefore extend the critic model from selecting among existing ideas to improving the idea-proposing policy itself. Here, the critic model serves as a learned reward model for GRPO training of the idea-proposing policy, assigning rewards to confident predictions and routing uncertain cases to empirical verification. This reduces the real-verification cost during training, enabling 2.75× as many policy updates under the same verification budget. In summary, our main contributions are below:

- We establish an effective training pipeline for idea-level critic models from real ML agent evolving trajectories. Through fine-tuning from soft-based reasoning critiques, our critic model outperforms both their base models and Gemini-3.1-Pro in static idea evaluation.
- We show that directly prompting an off-the-shelf frontier model, Gemini-3.1-Pro, as an idea selector yields only limited gains in OpenEvolve, whereas our trained critic models improve inference-time evolution under fixed verification budgets by concentrating empirical verification on the most promising proposals.
- We extend the use of critic models from idea selection to policy training by using their predictions as rewards and reserving empirical verification for uncertain cases. This reduces verification cost and enables more policy updates under a fixed verification budget.

2 BUILDING CRITERION-GUIDED IDEA-LEVEL CRITIC MODELS

2.1 TASK SETTING

Self-evolving ML agent framework. We consider an ML agent that iteratively improves a solution through proposal, implementation, and empirical verification. At round r, given a task t and a parent program P_r, the **idea proposer** generates a modification plan z_r. The **implementer** translates this

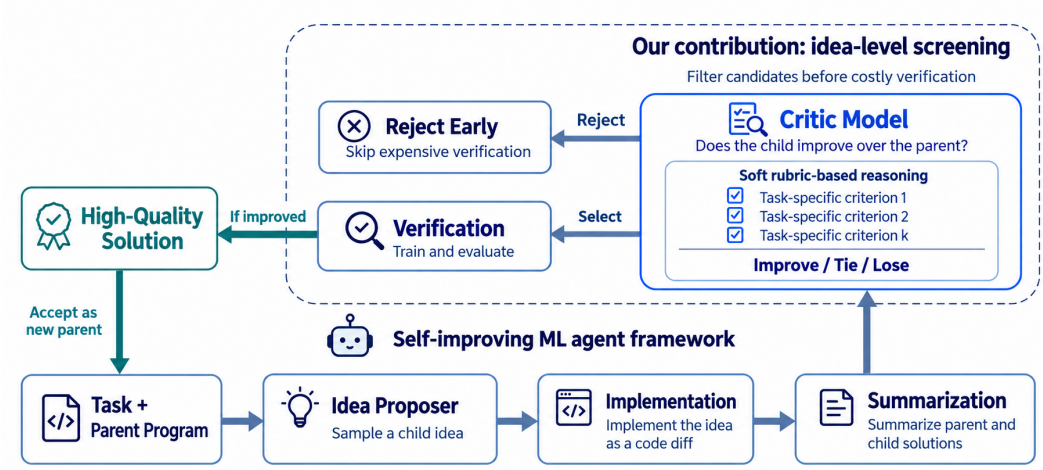


Figure 2: Overview of our idea-level screening pipeline. Given an ML task, a parent solution, and a candidate idea, the critic model predicts whether the idea will improve upon the parent. Only ideas predicted to improve are empirically verified, reducing costly verification.

plan into a child program C_r while preserving the proposed modification. An empirical evaluator E_t measures solution quality, and the agent retains the better program:

$$z_r = \text{Propose}(t, P_r), \qquad C_r = \text{Implement}(P_r, z_r), \tag{1}$$

and the parent is updated as child if child achieves better performance,

$$P_{r+1} = C_r \text{ if } E_t(C_r) > E_t(P_r) \text{ else } P_r. \tag{2}$$

Repeating this loop allows the agent to accumulate improvements, but verifying each candidate can require computationally costly model training and evaluation. We show an example of such framework in Appendix C.1.

Critic Model-Augmented ML Agent. Our goal is to predict the outcome of empirical verification from idea-level descriptions, enabling candidate screening before execution. To achieve this, we augment the agent pipeline with a critic model, as illustrated in Figure 2. To provide clear and neutral inputs, a **summarizer** converts the parent and child programs into concise descriptions p_r and a_r of their implemented approaches, removing exploratory reasoning, redundant details, and persuasive claims from the idea proposer's output. At a high level, the critic model parameterized by θ uses (t, p_r, a_r) to predict the execution-derived outcome:

$$\max_{\theta} \mathbb{E}_{(t,P_r,C_r)\sim\mathcal{D}} \left[\Pr_{\theta}(\text{Critic}_\theta(t, p_r, a_r) = y_r) \right], y_r = \begin{cases} \textsc{Improve}, & E_t(C_r) > E_t(P_r). \\ \textsc{Tie}, & E_t(C_r) = E_t(P_r). \\ \textsc{Lose}, & E_t(C_r) < E_t(P_r). \end{cases} \tag{3}$$

Here, $\mathcal{D}$ denotes the distribution of task–parent–child instances. These predictions allow the agent to prioritize promising candidates for empirical verification.

2.2 Criterion-Guided Critic Model Design

To make idea-level evaluation task-aware, we structure the critic model's reasoning around a set of task-specific criteria. The critic model must compare the parent and child solutions under each criterion before reaching a final verdict. For each ML task, we use Claude-4.8-Opus (Anthropic, 2026) in a multi-round probing process to analyze the task setup, data, evaluation metric, and modeling constraints. This process produces a detailed task understanding for task t and a fixed set of task-specific criteria, which are generated once and reused across all examples from the same task. Accordingly, each critic model input contains the task description, parent solution, proposed child solution, task understanding, and task-specific criteria, with example shown in Appendix C.2.

Soft-rubric reasoning. The critic model follows a soft-rubric reasoning template. It first analyzes every criterion and compares the parent and child solutions under that criterion, then freely determines

the relative importance of the criteria and aggregates its analyses into a final verdict. We call this design a *soft rubric* because the critic model must evaluate the fixed criteria but remains unrestricted in its analysis and weighting strategy. Appendix D and Appendix E provide the complete prompt instructions and a reasoning example, respectively.

2.3 Training Data and ML-Idea-Critic-Bench Construction

Dataset overview. The dataset contains approximately 28K execution-labeled examples: approximately 12.5K in the SFT/evaluation pool and 15.5K in the GRPO training pool. We divide the former into approximately 11K SFT training examples and 1.5K static evaluation examples, which constitute ML-Idea-Critic-Bench. The evaluation set contains roughly 500 examples from each of the three teacher-assistance strata defined below.

ML idea-evolution trajectories. We collect parent–child pairs from MLE-Dojo (Qiang et al., 2025) and MLS-Bench (Lyu et al., 2026), with Gemini-3.1-Pro generating both candidate ideas and their implementations. Initial parents are drawn from the baseline implementations provided by MLS-Bench or constructed as basic executable solutions for MLE-Dojo. Each trajectory example is represented as $x_i = (t_i, p_i, a_i, y_i)$, comprising a task description t_i, a parent idea p_i, a child idea a_i, and an outcome label y_i derived from execution. An execution first yields one of three outcomes: Improve, Tie, or Lose. To provide more precise targets for critique synthesis in the SFT/evaluation pool, we use the empirical distribution of child performance relative to the parent to further divide Improve into BigWin, MediumWin, and SmallWin, yielding the five-way label space

$$y_i \in \text{BigWin}, \text{MediumWin}, \text{SmallWin}, \text{Tie}, \text{Lose},$$

which captures both the direction and magnitude of the child's performance change relative to the parent. The GRPO pool retains the three-way outcome labels and is balanced across these classes by random downsampling. Detailed collection and label-construction are provided in Appendix B.

Teacher critique synthesis and difficulty strata. To collect high-quality SFT data, we use Gemini-3.1-Pro to generate soft-rubric reasoning critiques whose final predictions follow the same five-way label space. We first sample 8 critiques independently without revealing the execution-derived label y_i. An example is assigned to the *Sampled* stratum if at least one of these critiques predicts y_i correctly, in which case we retain one matching critique. If all 8 predictions are incorrect, we identify the most frequently predicted incorrect label, provide a hint explicitly ruling out that option, and sample another 8 critiques. An example is assigned to the *Revised* stratum if at least one of these new critiques predicts y_i correctly. If none succeeds, we reveal y_i and require Gemini-3.1-Pro to generate reasoning consistent with the correct outcome; these examples form the *Guided* stratum. The resulting SFT/evaluation pool contains three levels of increasing teacher assistance and example difficulty: *Sampled* (easy, **44.4%**), *Revised* (hard, **31.6%**), and *Guided* (harder, **23.9%**).

2.4 Critic Model Training

Supervised fine-tuning. We first fine-tune the critic model on approximately 11K Gemini-generated critiques collected using the procedure above, reserving the remaining 1.5K examples for static evaluation. By learning to produce both criterion-level analyses and final verdicts, the critic model learns the basic ability to evaluate ML proposals and follow the soft-rubric reasoning instructions.

Further Alignment with GRPO. Starting from the SFT checkpoint, we further train the critic model with Group Relative Policy Optimization (GRPO) (Shao et al., 2024) on an additional 15.5K execution-labeled examples not used for SFT. We use a binary reward: a rollout receives $+1$ if its predicted three-way verdict matches the execution-derived label and -1 otherwise. Details of training are presented in Appendix F.

3 ML-Idea-Critic Bench Result

Metrics. We report four complementary metrics. **Improve Accuracy** evaluates the binary decision of whether the child improves over its parent. **Dir-3** measures three-way accuracy over *Improve*, *Tie*,

Table 1: Idea-critic performance on the 1.5K-example evaluation set. Split-breakdown (Sampled/revised/guided) results are in Table 12. Best results are shown in bold.

Model	Training	Acc	Dir-3	F1	5-Ladder
Gemini-3.1-Pro	Zero-shot	54.3	46.6	59.8	28.2
Qwen-3.5-9B	Base	49.9	41.3	58.3	22.4
Qwen-3.5-35B	Base	49.9	36.4	51.8	20.1
Deepseek-GRM-16B	Ckpt from Liu et al. (2025)	58.5	57.8	73.3	25.3
Deepseek-GRM-27B	Ckpt from Liu et al. (2025)	58.4	58.1	73.6	25.3
Qwen-3.5-9B	SFT	62.4	55.6	70.9	31.5
Qwen-3.5-35B	SFT	65.4	60.3	74.1	38.5
Qwen-3.5-9B	SFT+GRPO	66.7	60.7	**75.6**	41.5
Qwen-3.5-35B	SFT+GRPO	**68.5**	**63.3**	75.4	**45.4**

and *Lose*. **F1** reports the F1 score for the binary improvement decision. Finally, **5-Ladder** evaluates the exact match accuracy over [BIGWIN > MEDIUMWIN > SMALLWIN > TIE > LOSE].

We compare our critics against Gemini-3.1-Pro, the base Qwen-3.5-9B and Qwen-3.5-35B models, and DeepSeek-GRM-16B/27B (Liu et al., 2025). DeepSeek-GRM is a generalist generative reward model trained with Self-Principled Critique Tuning to generate evaluation principles and critiques before assigning scores. We use the released DeepSeek-GRM-16B[1] and DeepSeek-GRM-27B[2] checkpoints, based on DeepSeek-V2-Lite and Gemma-2-27B, respectively. As shown in Table 1, SFT improves the Dir-3 accuracy of the 9B and 35B critics to 55.6% and 60.3%, surpassing Gemini-3.1-Pro (46.6%). GRPO further improves their accuracy to 60.7% and 63.3%, corresponding to gains of 5.1% and 3.0% over their SFT checkpoints and 14.1% and 16.7% over Gemini.

Ablation: Freestyle vs. Soft-Rubric Reasoning. The two reasoning formats perform similarly after SFT, with Dir-3 accuracies of 54.6% for freestyle reasoning and 55.6% for soft-rubric reasoning. Under GRPO, however, the freestyle critic deteriorates to 40.3%, whereas the soft-rubric critic improves to 60.7%. Qualitatively, the freestyle critic develops shorter analyses and superficial shortcuts, while the required criterion-by-criterion comparisons stabilize reinforcement learning. Results with full metrics can be found in Appendix G.

4 INTEGRATION WITH SELF-EVOLVING AGENTS

4.1 INFERENCE-TIME IDEA SELECTION

OpenEvolve setup. We integrate the critic model into OpenEvolve (Sharma, 2025) to screen candidate ideas before empirical verification. OpenEvolve is an evolutionary coding framework for recursive self-improvement that iteratively proposes, implements, and evaluates modifications as executable solutions. In this setup, Gemini-3.1-Pro serves as the idea proposer, implementer, and solution summarizer through three separate model calls. Each mutation follows the pipeline illustrated in Figure 2: children are compared with their parent at the idea level, and child idea is empirically verified only after the screening from critic model: The idea proposer Gemini-3.1-Pro is allowed to propose 16 ideas. For each idea, the critic model will make 16 reasoning critiques, and only the idea with the highest votes on win will be verified. For OpenEvolve setup, we maintain three islands and evolve each island for 10 rounds, yielding a total budget of 30 empirical verification runs per task. Additional details about the OpenEvolve configuration are provided in Appendix H.1. More details about critic model's idea-selection strategy are provided in Appendix H.2.

Screening settings and evaluation tasks. We compare seven settings that differ only in how candidate ideas are screened. *None* denotes vanilla OpenEvolve without idea-level screening. *Gemini-3.1-Pro* and *Gemini-3.5-Flash* use the corresponding pretrained models as critic models. *Base-9B*

[1] https://huggingface.co/BBQGOD/DeepSeek-GRM-16B
[2] https://huggingface.co/BBQGOD/DeepSeek-GRM-27B

Table 2: Openevolve with critic model integrated result with MLS-Lite-Bench with Gemini-3.1-Pro as idea proposer. All domains use 10 evolution rounds. The best result in each row is shown in **bold**.

Domain	None	Gemini		Qwen-3.5-9B		Qwen-3.5-35B	
		3.1-Pro	3.5-Flash	Base	Critic	Base	Critic
Language Models	18.04	18.85	17.66	32.48	**32.57**	32.54	31.47
Robotics	23.18	29.14	28.03	25.52	**29.75**	26.71	28.29
Vision and Generation	16.11	15.04	15.08	15.48	17.38	14.86	**17.84**
ML Systems and Efficient ML	64.00	64.01	63.73	63.11	**65.54**	63.59	63.80
AI for Science	33.55	**34.52**	32.38	33.94	31.67	32.68	33.09
Optimization and Theory	22.46	24.69	24.85	22.23	24.79	24.64	**24.86**
Classical and Adaptive Learning	51.85	51.26	51.50	45.25	**53.53**	45.43	47.40
Time Series and Forecasting	54.64	52.73	51.54	52.14	57.24	48.18	**58.72**
Structured and Causal Reasoning	55.81	54.35	55.93	52.83	54.01	52.61	**56.44**
Deep Learning	60.01	**64.90**	63.01	62.49	62.67	60.94	62.27
Trustworthy Learning	26.85	28.84	30.32	25.91	27.46	29.60	**30.99**
Domain Average	38.77	39.85	39.46	39.22	**41.51**	39.25	41.38
Heldout-Only Average	30.44	32.12	31.15	32.51	34.24	32.68	34.18

and *Base-35B* use the original Qwen-3.5-9B and Qwen-3.5-35B models, respectively, whereas *Critic-9B* and *Critic-35B* use our corresponding critic models trained with SFT and GRPO. We evaluate all settings on MLS-Lite-Bench, which contains 30 ML development tasks spanning 12 domains. Because every setting obtains a score of zero on `rl-value-discrete`, the benchmark's only reinforcement learning task, we exclude this non-discriminative task from aggregate reporting and present results over the remaining 29 tasks across 11 domains. The complete task configuration, domain grouping, and score normalization procedure are described in Appendix H.3.

Table 2 reports the MLS-Lite-Bench normalized score averaged within each domain. The settings using our trained critic models achieve the best score in 9 of the 11 domains and the highest overall average. This advantage persists on the 20 tasks excluded from critic model training: the 9B and 35B critic models achieve the two highest averages of 34.24 and 34.18, respectively, compared with 30.44 without screening. The largest gain appears in Language Models, where the normalized score increases from 18.04 without screening to 32.57 with the 9B critic model. Overall, the improvement margins are moderate because all settings share Gemini-3.1-Pro as the idea proposer and implementer.

4.2 Test Time Continual Learning

Table 3: Task-specific test-time continual learning result. TT stands for test time continual learning alignment. Scores higher than no continual learning baseline are shown in **bold**.

Task	Frozen Critic		TT-GRPO		TT-SDFT	
	9B	35B	9B	35B	9B	35B
ML Dimensionality Reduction	49.62	38.43	**50.06**	**42.15**	**50.43**	**51.77**
TS Exogenous Forecast	48.43	49.57	**48.90**	47.02	**49.11**	**51.17**
AI4Bio Mutation Effect Prediction	48.01	49.88	47.34	49.47	**51.66**	**50.34**
Graph Generation	70.69	73.30	**72.58**	73.05	**73.51**	**76.64**

We investigate whether adapting an idea-level critic during agent evolution improves the final solution quality under a fixed verification budget. We consider two strategies: test-time Group Relative Policy Optimization (GRPO) (Yuksekgonul et al., 2026) and Self-Distillation Fine-Tuning (SDFT) (Shenfeld et al., 2026). We pick 4 tasks from MLS-Lite-Bench to evaluate both strategies, under the same OpenEvolve setup in Section 4.1. Both methods start from our trained critic model checkpoint, keep its backbone parameters frozen, and optimize a LoRA adapter with rank 32. The updates use only

outcomes obtained from the empirical verifications already performed by OpenEvolve and therefore require no additional verifier calls.

Self-Distillation Fine-Tuning. Self-Distillation Fine-Tuning (SDFT) is introduced by Shenfeld et al. (2026) for continual learning from demonstrations. SDFT uses the same underlying model as both student and teacher, but provides the teacher with privileged context and distills its token-level predictions into the student along trajectories generated by the student itself. We adapt this idea to test-time alignment by treating the raw scores of parents and verified children as privileged teacher information. At every prefix of the student-generated critique, the teacher produces a next-token distribution, which is the fixed target for updating the student. We minimize the resulting token-level reverse-KL loss over the critique trajectory. We perform one SDFT update after every OpenEvolve round using the three newly verified parent–child pairs, one from each island.

Test-time GRPO. TTT-Discover (Yuksekgonul et al., 2026) formulated scientific discovery as test-time RL, in which a policy model updates its parameters using experience collected while solving the problem. Following this paradigm, we apply GRPO to the critic model during OpenEvolve. We perform one update every 2 evolution rounds, collecting 6 newly verified parent–child pairs across the three islands. For each pair, we sample 16 critic model rollouts and assign each rollout a reward of +1 if its predicted three-way outcome matches the empirical label and -1 otherwise.

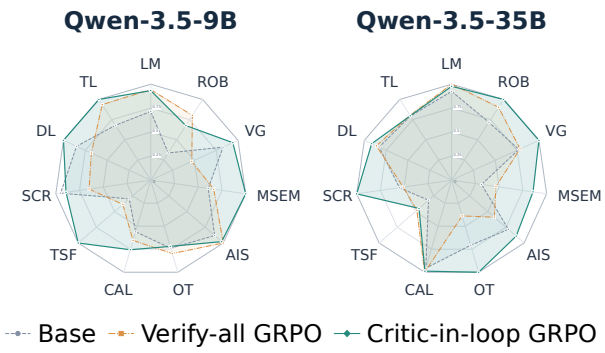
As shown in Table 3, SDFT improves downstream evolution more consistently than test-time GRPO. One possible explanation is the limited information in binary correctness rewards combined with the changing outcome distribution during evolution. When all 16 GRPO rollouts receive the same reward, their group-relative advantages vanish, providing no learning signal from the correctness reward. As the parent solution improves, successful modifications can also become rarer, encouraging the critic model to rely on the increasing prevalence of TIE/LOSE rather than distinguish promising candidates. Such behavior may achieve high prediction accuracy while providing little value for candidate selection. SDFT instead supplies outcome-conditioned token-level targets along the student's reasoning trajectory, offering a richer adaptation signal than verdict correctness alone. Our results align with existing work (Shenfeld et al., 2026) that this form of supervision can better support candidate selection and improve final solution quality under a limited verification budget.

5 THE CRITIC SERVING AS REWARD MODEL

While critic models help OpenEvolve select more promising ideas, its performance remains constrained by the proposal capabilities of the fixed policy model. We therefore investigate whether critic model predictions can provide useful training signals to improve the policy model's ability to generate better ideas.

5.1 GRPO WITH CRITIC-IN-LOOP

We further investigate whether the trained critic model can serve as a learned reward model for aligning the idea-proposing policy. We initialize the policy from the base Qwen-3.5-9B and Qwen-3.5-35B model and optimize it using GRPO (Shao et al., 2024) across the same 95 MLE-Dojo and 46 MLS-Bench training environments used for critic GRPO training. Each training step collects 16 parents, each with 16 rollout ideas, yielding 256 candidate ideas per step. The reward is binary: a solution receives $+1$ for an IMPROVE outcome and -1 otherwise. We consider two approaches, (1) baseline approach *verify-all*, where every candidate is empirically evaluated; and (2) *critic-in-loop* approach, where trained critic models serve as a reward model and replace real verification signals occasionally, depending on the critic's uncertainty on the candidate idea. Specifically, we use Critic-9B checkpoint serving base Qwen-3.5-9B and Critic-35B checkpoint serving base Qwen-3.5-35B respectfully. The critic models keep their parameters frozen throughout policy training. For each candidate a, let $v_y(a) = \frac{1}{16}\sum_{k=1}^{16} \mathbb{I}[\tilde{y}_k(a) = y]$ denote the fraction of critic model rollouts $\tilde{y}_k(a)$ predicting label y. We define uncertainty as $u(a) = 1 - \max_y v_y(a)$, namely, the fraction of rollouts that disagree with the majority. If at most three of the 16 rollouts disagree ($u(a) < 0.25$), we use the majority verdict to calculate reward; otherwise, we send the candidate to the ground-truth verifier.



Code	Domain
LM	Language Models
ROB	Robotics
VG	Vision and Generation
MSEM	ML Systems and Efficient ML
AIS	AI for Science
OT	Optimization and Theory
CAL	Classical and Adaptive Learning
TSF	Time Series and Forecasting
SCR	Structured and Causal Reasoning
DL	Deep Learning
TL	Trustworthy Learning

Figure 3: Trained idea-proposing policies performance on MLS-Lite-Bench. For Qwen-3.5-9B (left) and Qwen-3.5-35B (right), we compare the base policy, verify-all GRPO, and critic-model-in-the-loop GRPO across 11 domains. For each domain, scores are normalized against the best of the six arms.

5.2 Policy Model GRPO Result

For a controlled comparison, each method receives a total budget of 10,000 ground-truth verifications across all training environments. With 256 candidates per training step, *verify-all* exhausts this budget in approximately 40 steps. With critic-guided reward assignment, only 36.3% of policy-generated ideas require empirical verification, allowing approximately 110 steps under the same budget. This corresponds to a $2.75\times$ improvement in verification efficiency, measured as the number of candidate ideas assigned training rewards per ground-truth verification. This ratio does not measure wall-clock speedup or total compute savings, since critic inference incurs additional computation cost.

We evaluate the resulting policies directly on the MLS-Lite-Bench tasks. As shown in Figure 3, critic-in-loop GRPO improves the average normalized domain score from 19.57 to 25.21 for the 9B model and from 19.29 to 24.55 for the 35B model compared with verify-all GRPO. These results suggest that using critic-model predictions for confident cases allows the saved verification budget to support additional policy updates and thereby shifts the policy toward more promising ML modifications. Full numerical results are reported in Table 15 of Appendix I.1.

5.3 Ablation Experiments

Uncertainty Analysis. We analyze how the number of critic rollouts k and the uncertainty threshold T affect the reliability of majority-vote prediction. We evaluate how k and T influencing the following three quantities: the AUROC of the vote-based critic prediction, the fraction of critic errors recalled by uncertainty-based routing, and the fraction of all ideas sent to the verifier.

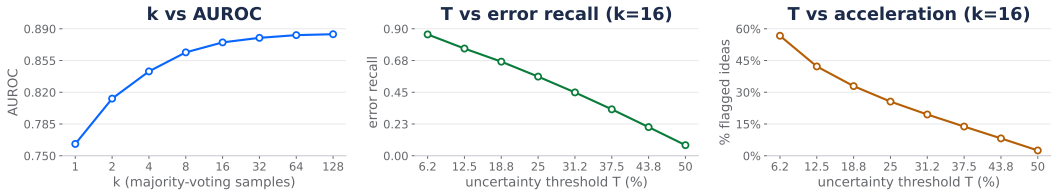


Figure 4: Uncertainty analysis. Increasing the number of critic samples yields diminishing AUROC gains beyond $k = 16$ (left). At $k = 16$, an uncertainty threshold of 0.25 recalls around 70% of incorrect predictions while routing about 30% of all ideas to empirical verification (middle and right).

Figure 4 reports the sweep on the test split of ML-Idea-Critic-Bench. Increasing k initially produces substantial improvements in the AUROC of the vote-based critic prediction, but the gains largely saturate at $k = 16$. With an uncertainty threshold around $T = 0.25$, the selective-verification rule identifies approximately 70% of the critic's errors while routing only about 30% of all ideas to the ground-truth verifier. This corresponds to our critic-in-loop experiments, 36.3% of policy-generated ideas are empirically verified, moderately exceeding the 30% estimated from the benchmark.

Ablation on the Critic as a Reward Model. We test whether critic model predictions alone can drive policy optimization. Starting from the base Qwen-3.5-35B policy, we perform GRPO using rewards provided by the frozen Critic-35B without real verification. We conduct two sweeps: varying the number of policy training steps from 0 to 160, and varying $k \in \{1, 4, 8, 16, 32\}$ while fixing training to 40 steps, where k is the number of critique rollouts for majority voting. Figure 5 in Appendix I.2 shows that policy performance generally improves as the number of training steps increases on two selected tasks. Increasing the number of critic model rollouts k also improves performance on the selected tasks, but the gains are comparatively modest and begin to saturate. In particular, increasing k beyond 16 provides limited additional benefit while proportionally increasing inference computational cost.

6 Related Work

Self-Evolving Agents and ML Agents. ML and self-evolving agents iteratively propose, implement, and evaluate solutions using empirical feedback (Huang et al., 2023; Jiang et al., 2025; Novikov et al., 2025; Sharma, 2025). Existing approaches improve this process through search and harness design with fixed pretrained models (Toledo et al., 2025; Yan et al., 2026b; Yamada et al., 2025), or by learning proposal policies from feedback (Zhang et al., 2026c; Yan et al., 2026a). Our work complements these directions by learning to evaluate proposed modifications before costly implementation and execution.

AI for Machine Learning Research. AI-assisted research systems coordinate literature review, experimentation, and manuscript preparation (Schmidgall et al., 2025; Tang et al., 2025), while related work studies automated paper reviewing (Tang et al., 2026; Liu et al., 2026a) and research-idea evaluation (Qiao et al., 2026a; Moussa et al., 2025). Closer to our setting, pretrained language model-based evaluation modules assess ML candidates and prioritize promising ones for execution (Foster et al., 2026; Zheng et al., 2026). We instead make the evaluator itself the primary learned component, training an idea-level critic model to predict the empirical value of proposed ML modifications.

Critic Models and Generative Reward Models. Conventional reward models typically assign scalar scores to model outputs (Ouyang et al., 2022; Lambert et al., 2025). Critic models and generative reward models produce textual assessments that make the basis of their judgments more explicit (Saunders et al., 2022; McAleese et al., 2024; Zhang et al., 2025; Chen et al., 2026b). These assessments support direct output evaluation (Zheng et al., 2023; Ke et al., 2024), actionable feedback for iterative refinement (Wang et al., 2023; Xie et al., 2025; Yang et al., 2025), and ranking among sampled candidates (Liu et al., 2025; Zhang et al., 2025; Saha et al., 2025). Rubric-guided approaches further structure evaluations across multiple criteria (Viswanathan et al., 2025; Gunjal et al., 2026; Guan et al., 2026; Qiu et al., 2026). Related work evaluates intermediate reasoning steps (Uesato et al., 2022; Lightman et al., 2024) or uses learned evaluations as reward signals for policy optimization and co-evolution (Shan et al., 2026; Li et al., 2026a; Chen et al., 2025; Zhang et al., 2026b). We focus on proposed modifications to existing ML solutions, whose ground-truth value requires empirical execution. Our critic model learns from empirical parent–child outcomes, supporting both inference-time candidate selection and policy training.

More discussion on related work is in Appendix A.

7 Conclusion

We introduced idea-level critic models as an efficient feedback mechanism for self-improving ML agents together with a training pipeline combining soft-rubric reasoning, SFT, and GRPO. Our critic model not only achieve strong performance in static idea evaluation, but also improve agents across inference-time idea selection, task-specific continual learning, and critic-guided policy optimization. By identifying promising ideas, the critic model enables agents to use limited verification budgets more effectively. As a learned reward model, the critic model also performs $2.75\times$ more policy-optimization steps under the same budget. Overall, our results establish idea-level critic model as a general and practical interface between idea generation and expensive execution, enabling more efficient ML agent learning and evolution.

AI USE STATEMENT

We used Gemini-3.1-Pro for synthetic data generation and Claude-4.8-Opus for task understanding and criterion construction. AI tools also assisted with debugging training code, preparing figures, and polishing author-written drafts. The authors developed the research questions, methods, and experimental design and interpreted the results. All AI-assisted work was reviewed by the authors, who take full responsibility for the paper and associated artifacts.

ETHICS STATEMENT

This work uses existing public benchmarks, MLE-Dojo and MLS-Bench, with model-generated trajectories and critiques and execution-derived outcome labels. It involves no new human-subject studies or manual data annotation. Selected solutions remain subject to empirical evaluation, and uncertain predictions are routed to empirical verification during policy training.

REPRODUCIBILITY STATEMENT

We document data collection and label construction in Appendix B, training configurations and hyperparameters in Appendix F, and prompt templates in Appendix D. The main text describes the critic model training pipeline and evaluation procedures. We will publicly release our code and generated datasets to support further research.

REFERENCES

Anthropic. Claude opus (version 4.8), 2026. URL https://www.anthropic.com/news/claude-opus-4-8.

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. In *International Conference on Learning Representations*, volume 2025, pages 50466–50494, 2025.

Yuan Chang, Ziyue Li, Hengyuan Zhang, Yuanbo Kong, Yanru Wu, Hayden Kwok-Hay So, Zhijiang Guo, Liya Zhu, and Ngai Wong. Treereview: A dynamic tree of questions framework for deep and efficient llm-based scientific peer review. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 15662–15693, 2025.

Jiaqi Chen, Bang Zhang, Ruotian Ma, Peisong Wang, Xiaodan Liang, Zhaopeng Tu, Xiaolong Li, and Kwan-Yee K Wong. Spc: Evolving self-play critic via adversarial games for llm reasoning. *Advances in Neural Information Processing Systems*, 2025.

Xi Chen, Jie Mu, Mo Xuan, and Qun Shao. Graph-structured rubrics: Compiling rubrics into typed evaluation graphs for llm judges. *arXiv preprint arXiv:2608.12097*, 2026a.

Xiusi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru Wang, Yu Zhang, Denghui Zhang, Tong Zhang, et al. Rm-r1: Reward modeling as reasoning. In *International Conference on Learning Representations*, volume 2026, pages 88313–88342, 2026b.

Mike D'Arcy, Tom Hope, Larry Birnbaum, and Doug Downey. Marg: Multi-agent review generation for scientific papers. *arXiv preprint arXiv:2401.04259*, 2024.

Damon Falck, Samer Sabri, Anja Surina, Thom Foster, Anya Sims, Sam Devlin, Dylan Rogers, Tantum Collins, Kaloyan Aleksiev, Louis Kirsch, et al. Training ai scientists to replicate research. *arXiv preprint arXiv:2608.13331*, 2026.

Thomas Simon Foster, Bassel Al Omari, Tingchen Fu, Thomas Mann, Carl Domond, Lucia Cipolina-Kun, Bhavul Gauri, Muna Aghamelu, Alexander D Goldie, Eryk Helenowski, et al. Ai research preference models. *arXiv preprint arXiv:2608.13940*, 2026.

Xian Gao, Jiacheng Ruan, Zongyun Zhang, Jingsheng Gao, Ting Liu, and Yuzhuo Fu. Reviewagents: Bridging the gap between human and ai-generated paper reviews. *arXiv preprint arXiv:2503.08506*, 2025.

Zhaolin Gao, Kianté Brantley, and Thorsten Joachims. Reviewer2: Optimizing review generation through prompt generation. *arXiv preprint arXiv:2402.10886*, 2024.

Google Gemini Team. Gemini 3.1 Pro: A smarter model for your most complex tasks, February 2026. URL https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/.

Xin Guan, Xiaomeng Hu, Shen Huang, Zhenyi Wang, Bo Zhang, Zijian Li, Pengjun Xie, Bo Liu, and Jiuxin Cao. Evorubric: Self-evolving rubric-driven rl for open-ended generation. *arXiv preprint arXiv:2605.29847*, 2026.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *International Conference on Learning Representations*, volume 2026, pages 127924–127945, 2026.

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.

Can Jin, Jiakang Li, Rui Wu, Eddy Z Zhang, and Dimitris N Metaxas. Weak critics make strong learners: On-policy critique distillation for scalable oversight. *arXiv preprint arXiv:2606.00424*, 2026.

Pei Ke, Bosi Wen, Andrew Feng, Xiao Liu, Xuanyu Lei, Jiale Cheng, Shengyuan Wang, Aohan Zeng, Yuxiao Dong, Hongning Wang, et al. Critiquellm: Towards an informative critique generation model for evaluation of large language model generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13034–13054, 2024.

Sunnie SY Kim, Wesley Hanwen Deng, Jennifer Wortman Vaughan, Buxin Su, Weijie Su, Alekh Agarwal, Sharon Li, Martin Jaggi, Daniel G Goldstein, Nihar B Shah, et al. Use and effects of llms in peer review: A randomized experiment and survey at icml 2026. *arXiv preprint arXiv:2609.19420*, 2026.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. Rewardbench: Evaluating reward models for language modeling. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1755–1797, 2025.

Ruochen Li, Teerth Patel, Qingyun Wang, and Xinya Du. Mlr-copilot: Autonomous machine learning research based on large language models agents. *arXiv preprint arXiv:2408.14033*, 2024.

Zhicong Li, Lingjie Jiang, Yulan Hu, Xingchen Zeng, Yixia Li, Xiangwen Zhang, Guanhua Chen, Zheng Pan, Xin Li, and Yong Liu. No more stale feedback: Co-evolving critics for open-world agent learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12643–12660, 2026a.

Zhuofeng Li, Yi Lu, Dongfu Jiang, Haoxiang Zhang, Yuyang Bai, Chuan Li, Yu Wang, Shuiwang Ji, Jianwen Xie, and Yu Zhang. Reviewgrounder: Improving review substantiveness with rubric-guided, tool-integrated agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32015–32044, 2026b.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601, 2024.

Jiaqi Liu, Shi Qiu, Mairui Li, Bingzhou Li, Haonian Ji, Siwei Han, Xinyu Ye, Peng Xia, Zihan Dong, Meng Chen, et al. Autoresearchclaw: Self-reinforcing autonomous research with human-ai collaboration. *arXiv preprint arXiv:2605.20025*, 2026a.

Yaowenqi Liu, BingXu Meng, Rui Pan, Yuxing Liu, Jerry Huang, Jiaxuan You, and Tong Zhang. Guide: towards scalable advising for research ideas. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 42813–42834, 2026b.

Zijun Liu, Kaiming Liu, Yiqi Zhu, Xuanyu Lei, Zonghan Yang, Zhenhe Zhang, Peng Li, and Yang Liu. Aigs: Generating science from ai-powered automated falsification. *arXiv preprint arXiv:2411.11910*, 2024.

Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. Inference-time scaling for generalist reward modeling. *arXiv preprint arXiv:2504.02495*, 2025.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

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

Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. Llm critics help catch llm bugs. *arXiv preprint arXiv:2407.00215*, 2024.

Hanane Nour Moussa, Patrick Queiroz Da Silva, Daniel Adu-Ampratwum, Alyson East, Zitong Lu, Nikki Puccetti, Mingyi Xue, Huan Sun, Bodhisattwa Prasad Majumder, and Sachin Kumar. Scholareval: Research idea evaluation grounded in literature. *arXiv preprint arXiv:2510.16234*, 2025.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Arik, and Tomas Pfister. Mle-star: Machine learning engineering agent via search and targeted refinement. *Advances in Neural Information Processing Systems*, 38:116692–116712, 2025.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

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

Shuofei Qiao, Yunxiang Wei, Xuehai Wang, Bin Wu, Boyang Xue, Ningyu Zhang, Hossein A Rahmani, Yanshan Wang, Qiang Zhang, Keyan Ding, et al. Innoeval: On research idea evaluation as a knowledge-grounded, multi-perspective reasoning problem. *arXiv preprint arXiv:2602.14367*, 2026a.

Shuofei Qiao, Yanqiu Zhao, Zhisong Qiu, Xiaobin Wang, Jintian Zhang, Zhao Bin, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, et al. Scaling generalist data-analytic agents. In *International Conference on Learning Representations*, volume 2026, pages 14942–14971, 2026b.

Weijie Qiu, Dai Guan, Junxin Wang, Zhihang Li, Yongbo Gai, Mengyu Zhou, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Rationale matters: Learning transferable rubrics via proxy-guided critique for vlm reward models. *arXiv preprint arXiv:2603.16600*, 2026.

Swarnadeep Saha, Xian Li, Marjan Ghazvininejad, Jason Weston, and Tianlu Wang. Learning to plan & reason for evaluation with thinking-llm-as-a-judge. *arXiv preprint arXiv:2501.18099*, 2025.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*, 2022.

Samuel Schmidgall and Michael Moor. Agentrxiv: Towards collaborative autonomous research. *arXiv preprint arXiv:2503.18102*, 2025.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using llm agents as research assistants. *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 5977–6043, 2025.

Zikang Shan, Han Zhong, Liwei Wang, and Li Zhao. Bringing value models back: Generative critics for value modeling in llm reinforcement learning. *arXiv preprint arXiv:2604.10701*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024.

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025. URL `https://github.com/algorithmicsuperintelligence/openevolve`.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. Ai-researcher: Autonomous scientific innovation. *Advances in Neural Information Processing Systems*, 38:9481–9520, 2025.

Qiong Tang, Tianxiang Sun, Xiangkun Hu, Xiangyang Liu, Yiran Chen, Yunfan Shao, Bobo Li, Changze Lv, Cheng Xu, Chengsong Huang, et al. Fars: A fully automated research system deployed at scale. *arXiv preprint arXiv:2606.31651*, 2026.

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Lupidi, et al. Ai research agents for machine learning: Search, exploration, and generalization in mle-bench. *Advances in Neural Information Processing Systems*, 38:35309–35348, 2025.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Vijay Viswanathan, Yanchao Sun, Xiang Kong, Meng Cao, Graham Neubig, and Tongshuang Wu. Checklists are better than reward models for aligning language models. *Advances in Neural Information Processing Systems*, 2025.

Beining Wang, Weihang Su, Hongtao Tian, Hao Kong, Tao Yang, Ting Yao, Qingyi Pan, Yueyue Wu, Qingyao Ai, Min Zhang, et al. Co-evolving llm evaluators and policies via dynamicrubric. *arXiv preprint arXiv:2607.20083*, 2026a.

Chenrun Wang, Mingxuan Zhu, Tiancheng Huang, Wenjie Li, Yujie Zhang, Zichen Zhu, Zhiying Zou, Kai Yu, and Lu Chen. Ligbench: A unified and human-aligned benchmark for llm-based research idea generation. *arXiv preprint arXiv:2608.13136*, 2026b.

Jianze Wang, Kunwang Zheng, Ying Liu, Yu Cao, Qilong Zhang, Jinlong Chen, Hua Yang, and Qianglong Chen. Serpo: Self-evolving rubric policy optimization for open-ended test-time reinforcement learning. *arXiv preprint arXiv:2607.26873*, 2026c.

Tianlu Wang, Ping Yu, Xiaoqing Ellen Tan, Sean O'Brien, Ramakanth Pasunuru, Jane Dwivedi-Yu, Olga Golovneva, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. Shepherd: A critic for language model generation. *arXiv preprint arXiv:2308.04592*, 2023.

Yixuan Weng, Minjun Zhu, Guangsheng Bao, Hongbo Zhang, Jindong Wang, Yue Zhang, and Linyi Yang. Cycleresearcher: Improving automated research via automated review. In *International Conference on Learning Representations*, volume 2025, pages 3669–3709, 2025.

Yixuan Weng, Minjun Zhu, Qiujie Xie, Qiyao Sun, Zhen Lin, Sifan Liu, and Yue Zhang. Deepscientist: Advancing frontier-pushing scientific findings progressively. In *International Conference on Learning Representations*, volume 2026, pages 47981–48037, 2026.

Zhihui Xie, Jie Chen, Liyu Chen, Weichao Mao, Jingjing Xu, and Lingpeng Kong. Teaching language models to critique via reinforcement learning. *arXiv preprint arXiv:2502.03492*, 2025.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

Minghao Yan, Bo Peng, Benjamin Coleman, Ziqi Chen, Zhouhang Xie, Shuo Chen, Zhankui He, Noveen Sachdeva, Weili Wang, Ed H Chi, et al. Pacevolve++: Improving test-time learning for evolutionary search agents. *arXiv preprint arXiv:2605.07039*, 2026a.

Minghao Yan, Bo Peng, Benjamin Coleman, Ziqi Chen, Zhouhang Xie, Shuo Chen, Zhankui He, Noveen Sachdeva, Isabella Ye, Weili Wang, et al. Pacevolve: Enabling long-horizon progress-aware consistent evolution. *arXiv preprint arXiv:2601.10657*, 2026b.

Ruihan Yang, Fanghua Ye, Jian Li, Siyu Yuan, Zhaopeng Tu, Xiaolong Li, Deqing Yang, et al. The lighthouse of language: Enhancing llm agents via critique-guided improvement. *Advances in Neural Information Processing Systems*, 2025.

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

Ya-Qi Yu, Hao Wang, Fangyu Hong, Xiangyang Qu, Gaojie Wu, Qiaoyu Luo, Nuo Xu, Huixin Wang, Wuheng Xu, Yongxin Liao, et al. Reinforcement learning with robust rubric rewards. *arXiv preprint arXiv:2605.30244*, 2026.

Zhaojian Yu, Kaiyue Feng, Yilun Zhao, Shilin He, Xiao-Ping Zhang, and Arman Cohan. Alpharesearch: Accelerating new algorithm discovery with language models. *arXiv preprint arXiv:2511.08522*, 2025.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, et al. Learning to discover at test time. *arXiv preprint arXiv:2601.16175*, 2026.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pages 104223–104294, 2026a.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Seyed Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *International Conference on Learning Representations*, volume 2025, pages 12476–12505, 2025.

Yudi Zhang, Meng Fang, Zhenfang Chen, and Mykola Pechenizkiy. Self-evolving llm agents with in-distribution optimization. *arXiv preprint arXiv:2606.07367*, 2026b.

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

Jingsheng Zheng, Jintian Zhang, Yujie Luo, Yuren Mao, Yunjun Gao, Lun Du, Huajun Chen, and Ningyu Zhang. Can we predict before executing machine learning agents? In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3941–3974, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

8 APPENDIX

A MORE RELATED WORK

Self-Evolving Agents and ML Agents. ML agents automate ML development by translating task specifications into executable solutions, running training and evaluation pipelines, interpreting empirical results, and iteratively revising their approaches (Huang et al., 2023; Chan et al., 2025; Qiang et al., 2025; Jiang et al., 2025; Qiao et al., 2026b; Yu et al., 2025). Closely related self-evolving and recursively self-improving agents repeatedly modify solutions using feedback from previous iterations (Lu et al., 2024; Zhang et al., 2026a; Novikov et al., 2025; Sharma, 2025). These systems commonly follow a propose–implement–execute–refine loop, and recent work increasingly targets the generation and selection of promising candidates before execution. Some approaches improve the surrounding agent harness while keeping the pretrained model fixed, using structured search and refinement to explore stronger solutions (Toledo et al., 2025; Nam et al., 2025; Yan et al., 2026b; Yamada et al., 2025); others learn proposal policies directly from feedback (Zhang et al., 2026c; Yan et al., 2026a).

AI for Machine Learning Research. LLM-based agents increasingly automate machine learning research, from literature review and ideation to experimentation, analysis, and manuscript preparation (Lu et al., 2024; Schmidgall et al., 2025; Tang et al., 2025). Related systems connect idea generation to execution, introduce falsification, and support collaborative or sustained research (Li et al., 2024; Liu et al., 2024; Schmidgall and Moor, 2025; Weng et al., 2026). Recent work further explores research automation at scale and iterative improvement through human feedback and accumulated experience (Tang et al., 2026; Liu et al., 2026a). A complementary line develops AI systems for paper reviewing through multi-agent feedback, aspect-conditioned generation, structured questioning, and rubric-guided evaluation (D'Arcy et al., 2024; Gao et al., 2024; 2025; Chang et al., 2025; Li et al., 2026b). Learned reviewing models also support iterative refinement of research outputs (Weng et al., 2025), while empirical studies examine the effects of LLM use in peer review (Kim et al., 2026). Closer to our setting, pretrained language model-based evaluation modules assess unverified ideas or candidate ML solutions to prioritize empirical execution (Weng et al., 2026; Zheng et al., 2026; Foster et al., 2026). Our work focuses on training an execution-grounded idea-level critic model to predict the empirical value of proposed ML modifications. The critic model evaluates each modification relative to an existing solution and learns from outcome labels derived from empirical parent–child comparisons. We develop a training recipe for this prediction task and apply the resulting critic model to candidate selection during inference-time evolution and reward provision during policy optimization.

Critic Models and General Reward Models. Conventional reward models compress the quality of a model output into a scalar score, providing limited insight and often struggling with open-ended problems whose success depends on multiple interacting factors (Ouyang et al., 2022; Lambert et al., 2025). Critic models and general reward models instead generate reasoning-style analyses before reaching a final judgment (Yao et al., 2022), making their evaluations more interpretable and allowing them to produce more reliable, task-general feedback (Liu et al., 2025; Saunders et al., 2022; Zheng et al., 2023; McAleese et al., 2024; Zhang et al., 2025; Chen et al., 2026b). Existing work uses such generative evaluators in several related ways. At inference time, they can directly assess completed outputs (Zheng et al., 2023; Ke et al., 2024), provide actionable feedback for iterative refinement (Saunders et al., 2022; McAleese et al., 2024; Wang et al., 2023; Xie et al., 2025; Yang et al., 2025; Jin et al., 2026), or rank multiple sampled candidates so that only the most promising response is returned (Liu et al., 2025; Zhang et al., 2025; Saha et al., 2025). In structured scenarios, rubric-guided critiques further regularize multi-dimensional evaluations and mitigate evaluation degradation (Liu et al., 2025; Viswanathan et al., 2025; Gunjal et al., 2026; Wang et al., 2026a; Guan et al., 2026; Wang et al., 2026c; Qiu et al., 2026; Chen et al., 2026a; Yu et al., 2026). Other approaches formulate generative critics more explicitly as value models: they estimate

the quality or expected return of intermediate states and trajectories (Uesato et al., 2022; Lightman et al., 2024), and their judgments can be converted into learning signals for policy optimization, including PPO-style training and adversarial co-evolution (Shan et al., 2026; Wang et al., 2026a; Li et al., 2026a; Chen et al., 2025; Zhang et al., 2026b). These methods have primarily been developed to evaluate language or reasoning outputs produced by policy models. A separate line of work evaluates research idea (Qiao et al., 2026a; Moussa et al., 2025; Falck et al., 2026; Liu et al., 2026b; Wang et al., 2026b) by constructing methodology on human level ideas, which are more tailored to AutoResearch and human-AI collaboration research, which is beyond the scope of our paper.

B Data Collection and Label Construction

B.1 Task Environments and Initial Parent Solutions

We construct our data from selected task environments in MLE-Dojo (Qiang et al., 2025) and MLS-Bench (Lyu et al., 2026). MLS-Bench provides source code and baseline implementations for each task, which we directly use as the initial parent solutions. MLE-Dojo provides executable task environments but does not include comparable starter solutions. For each selected MLE-Dojo task, we therefore prompt Gemini-3.1-Pro to generate a basic runnable solution, execute it in the corresponding environment, and use its resulting score as the initial parent score.

B.2 Trajectory Collection and Label Construction

SFT/evaluation pool: two-stage collection. We construct the SFT/evaluation pool using a two-stage procedure. In the first stage, we run ten sequential evolution steps on each selected task, starting from its initial executable solution. The solution produced at each step is retained as a parent for the second stage. In the second stage, we independently propose and execute 20 child solutions for each retained parent, with all 20 children evaluated against the same parent. Gemini-3.1-Pro generates the ideas and implements the solutions in both stages. The resulting parent–child pairs form the candidate pool for SFT and static evaluation.

Parent-specific labels. For each parent p, we compute the signed relative improvements of its executed children, orienting the task metric so that positive values indicate better performance than the parent. We exclude infeasible child implementations and parents with fewer than three improving children. The child outcomes associated with each retained parent provide the empirical mean μ_p and standard deviation σ_p used to construct the five-level labels.

GRPO pool: improvement-driven collection. We collect GRPO training examples using a separate iterative procedure across the designated training tasks. At each step, Gemini-3.1-Pro proposes and implements one child solution from the current parent. If empirical evaluation shows an improvement, the child becomes the parent for the next step; otherwise, the current parent is retained. This procedure collects examples across a changing set of parents as successful modifications accumulate.

The initial GRPO pool contains approximately 25K execution-labeled examples, with an imbalanced distribution of Improve, Tie, and Lose outcomes. We randomly downsample within outcome classes to obtain a balanced training set of 15,669 examples, comprising 5,223 examples per class, as imbalanced distribution will cause critic model GRPO training to always output the majority label. This pool is used for critic GRPO training and is separate from the SFT/evaluation pool.

B.3 Parent-Specific Label Thresholds, Dataset Statistics, and Label Distribution

The same absolute improvement can have different significance depending on the quality and difficulty of the current parent solution. We therefore construct parent-specific labels. For each parent p, we generate and execute 20 child solutions and compute their signed relative improvements $\Delta_{p,1}, \ldots, \Delta_{p,20}$, orienting each task metric so that a positive value always indicates improvement. We filter out parents that have less than 3 improved children solutions. We approximate the resulting improvement distribution for each parent with a Gaussian whose empirical mean and standard

deviation are denoted by μ_p and σ_p. The standardized improvement of child j is

$$z_{p,j} = \frac{\Delta_{p,j} - \mu_p}{\sigma_p}.$$

We assign the five-level label in descending order, using the first condition that applies:

$$y_{p,j} = \begin{cases} \text{BIGWIN}, & \Delta_{p,j} > 0 \text{ and } (\Delta_{p,j} > 0.25 \text{ or } z_{p,j} > 2), \\ \text{MEDIUMWIN}, & \Delta_{p,j} > 0 \text{ and } (\Delta_{p,j} > 0.10 \text{ or } z_{p,j} > 1), \\ \text{SMALLWIN}, & \Delta_{p,j} > 0 \text{ and } z_{p,j} > 0.2, \\ \text{TIE}, & \Delta_{p,j} \geq 0 \text{ or } -0.2 \leq z_{p,j} \leq 0.2, \\ \text{LOSE}, & \text{otherwise.} \end{cases}$$

The absolute relative-improvement thresholds identify large gains even for parents whose child scores have high variance, while the standardized thresholds adapt the labels to the local difficulty of improving each parent. The Gaussian approximation is used only to compute the parent-specific standardized score; the final labels are assigned deterministically using the rules above.

After removing infeasible child solutions, the resulting SFT dataset. The five labels in SFT dataset account for **25.4%** BIGWIN, **17.5%** MEDIUMWIN, **15.5%** SMALLWIN, **13.6%** TIE, and **28.0%** LOSE.

C TEMPLATES AND OUTPUTS EXAMPLES

C.1 ML AGENT PIPELINE EXAMPLES

We present a recorded example of the ML agent pipeline on the `dl-activation-function` task from MLS-Lite-Bench. The example follows one candidate from idea proposal through implementation, summarization, critic model screening, and empirical verification. The tables retain the recorded prompts and generated outputs, with repeated inputs referenced across tables.

Task and parent solution. Table 4 presents the task description and the initial parent program. The task is to design a custom activation function while keeping the model architectures, optimizer, and data pipeline fixed. The parent uses ReLU and achieves a `combined_score` of 0.5997.

Idea proposal. Table 5 shows the instructions given to the idea proposer and its generated approach. Conditioned on the task and parent program, the proposer suggests Mish-ACON-C, a Mish-based activation with bounded learnable parameters that control its slopes and transition sharpness.

Implementation. Table 6 shows how the committed idea is translated into executable code. The implementer receives the proposed approach and the parent program, and replaces only the editable `CustomActivation` class. This implementation is produced before critic model screening and is reused if the candidate is selected for empirical verification.

Solution summarization. Table 7 presents the summarization instructions and the resulting child description. The summarizer receives both the proposed approach and its implementation, and is instructed to describe what the code does without predicting its performance. The resulting code-grounded summary serves as the child input to the critic model.

Table 4: Running example: task context and parent program for task activation-function design.

Task and parent program

Research Question

Design an activation function for deep convolutional neural networks that improves test accuracy across different architectures (ResNet, VGG) and datasets (CIFAR-10, CIFAR-100, FashionMNIST), while keeping the model definitions, optimizer, initialization, and data pipeline fixed.

Background

Activation functions introduce nonlinearity into neural networks and critically affect training dynamics, gradient flow, sparsity, and generalization. Classic and modern choices include:

- **ReLU**: `max(0, x)` — simple, sparse, but zero gradient for negative inputs .

- **GELU**: `x * Phi(x)` where `Phi` is the standard Gaussian CDF.

- **Swish / SiLU**: `x * sigmoid(beta * x)`; self-gated, smooth, non-monotonic.

- **Mish**: `x * tanh(softplus(x))`; self-regularized, smooth, non-monotonic.

- **Squared ReLU**, **StarReLU**, and other variants explore polynomial gates and learnable/affine extensions.

What You Can Modify

The `CustomActivation` class inside `pytorch-vision/custom_activation.py`. It is an `nn.Module` used as a drop-in replacement for ReLU throughout the network.

You may modify the `forward` computation (any element-wise or channel-wise operation), register learnable parameters in `__init__`, choose any shape of activation curve (monotonic / non-monotonic / bounded), and decide negative-domain behavior (zero, linear, bounded, learnable). Tensor shape must be preserved.

The activation is used in:

- ResNet: BasicBlock (twice per block) and the initial conv.

- VGG: after every Conv-BN pair and inside the classifier head.

- MobileNetV2: replaces the ReLU6 baseline used in inverted residuals.

Fixed Pipeline

- Optimizer: SGD with `lr=0.1`, `momentum=0.9`, `weight_decay=5e-4`.

- Schedule: cosine annealing over `200` epochs.

- Data augmentation: `RandomCrop(32, pad=4)` + `RandomHorizontalFlip`.

- Weight initialization: standard Kaiming normal (fixed).

- Evaluation settings: ResNet-20 on CIFAR-10, VGG-16-BN on CIFAR-100, MobileNetV2 on FashionMNIST.

Baselines

- **gelu** — Hendrycks & Gimpel, arXiv:1606.08415; `nn.GELU` (no learnable parameters).

- **silu** — Ramachandran et al. / Elfwing et al., arXiv:1710.05941; `nn.SiLU`, equivalent to Swish with `beta=1` (no learnable parameters).

- **mish** — Misra, arXiv:1908.08681; `x * tanh(softplus(x))` (no learnable parameters).

Metric

Best test accuracy (%, higher is better) achieved during training. The activation must be differentiable, shape-preserving, and must not change normalization layers, residual blocks, classifier heads, datasets, or the training loop.

Parent editable band (source Section 0.2)

```
class CustomActivation(nn.Module):
    """Custom activation function. Drop-in replacement for ReLU.
    Input/Output: tensor of any shape, element-wise operation.
    Design considerations:
        - Monotonicity vs non-monotonicity tradeoffs
        - Smoothness (differentiability everywhere vs piecewise)
        - Negative-domain behavior (zero, linear, bounded)
        - Computational cost (simple ops vs learned parameters)
        - Interaction with BatchNorm and residual connections
    """

    def __init__(self):
        super().__init__()

    def forward(self, x):
        return F.relu(x)  # default: ReLU
```

Table 5: Idea proposal for the activation-function example. The system instruction, stage-specific user instruction, and recorded approach are shown. Repeated task and parent inputs are referenced by table number.

Stage 1: idea proposal

System prompt

You are an expert ML researcher proposing ONE concrete improvement to an existing solution. You propose the IDEA only — you do NOT write code in this step. Be specific and technical: name the modeling/training/feature change and why it should move the target metric. Propose a single coherent idea, not a menu of options.

User prompt

Task: mls__dl-activation-function

Shared input: insert the complete task description from Table 4.

Metric: **combined_score** (higher is better)

Current parent solution (current parent verifier score: 0.5997 — beat it)

Shared input: insert the parent editable band from Table 4.

Inspiration: an alternative approach (verifier score: 0.5997)

A neutral summary of a DIFFERENT approach that also scored well. Use it as a source of ideas to combine/contrast/diverge from — do NOT copy it verbatim. (baseline)

Your task

Propose ONE concrete, self-contained improvement idea over the parent solution above that should improve the metric (higher is better). It must be implementable as an edit to the parent within the runtime budget, using only standard ML libraries and the data already present — do NOT invent data, files, columns, or external resources. Emit EXACTLY two sections and nothing else:

Approach

The concrete change(s) you propose and the mechanism by which they should improve the metric. Be specific (architecture/algorithm/hyperparameters/features/loss/validation), not vague.>

Proposed approach (actual model output, source Section 1.3)

We propose **Mish-ACON-C**, a parameterized activation function that merges the Activate Or Not (ACON) framework with the Mish activation to learn layer-specific non-linearities. The function computes `diff = (p1 - p2)* x` and outputs `diff * torch.tanh(F.softplus(beta * diff))+ p2 * x`, governed by three learnable scalar parameters per layer: positive slope (`p1`), negative slope (`p2`), and transition sharpness (`beta`). To guarantee stability under the fixed SGD optimizer and prevent weight decay from collapsing the slopes, these scalars are formulated as zero-initialized offsets bounded by a `tanh` squash (e.g., `p1 = 1.0 + 0.5 * torch.tanh(p1_offset),p2 = 0.25 * torch.tanh(p2_offset), beta = 1.0 + 0.8 * torch.tanh(beta_offset)`). By defaulting exactly to the state-of-the-art Mish baseline at initialization, this mechanism strictly expands the function space safely, allowing the network to dynamically tune its gating and asymptotes to improve test accuracy without risking gradient explosion.

Table 6: Implementation of the committed Mish-ACON-C idea. The replacement code is produced before critic model screening and is subsequently reused for verification. Repeated inputs and the full fixed code context are referenced by table number.

Stage 2: implementation

System prompt

You are an expert ML researcher. Implement the committed idea by rewriting ONLY the editable band of the file. Emit the replacement band in a single "`python`" block and nothing else after it.

User prompt

Committed idea (implement EXACTLY this)

You have already decided on the following idea. Implement EXACTLY this idea. Do NOT change the approach, do NOT substitute a different method, do NOT add unrelated changes — just turn this idea into correct, runnable code.

Shared input: insert the complete proposed approach from Table 5.

Task: dl-activation-function

Shared input: insert the complete task description from Table 4.

Editable contract

You may ONLY replace the EDITABLE band (lines 32-49) of `pytorch-vision/custom_activation.py`. Everything OUTSIDE that band (the FIXED header/footer shown below, and any read-only sibling files) is IMMUTABLE — the official verifier byte-compares it against the reference and scores 0 if it changed. Your replacement must keep the SAME public interface (function/class/variable names, signatures) that the FIXED code imports or calls.

FIXED header (read-only, lines 1-31)

Shared input: insert the fixed header from Table **??**.

FIXED footer (read-only, lines 50+)

Shared input: insert the fixed footer from Table **??**.

Current editable band (lines 32-49)

Shared input: insert the parent editable band from Table 4.

Return the FULL replacement for this band (a drop-in for lines 32-49) in one "`python`" block.

Implemented child code

```
class CustomActivation(nn.Module):
    """Custom activation function. Drop-in replacement for ReLU.

    Input/Output: tensor of any shape, element-wise operation.

    Design considerations:
        - Monotonicity vs non-monotonicity tradeoffs
        - Smoothness (differentiability everywhere vs piecewise)
        - Negative-domain behavior (zero, linear, bounded)
        - Computational cost (simple ops vs learned parameters)
        - Interaction with BatchNorm and residual connections
    """

    def __init__(self):
        super().__init__()
        self.p1_offset = nn.Parameter(torch.zeros(1))
        self.p2_offset = nn.Parameter(torch.zeros(1))
        self.beta_offset = nn.Parameter(torch.zeros(1))

    def forward(self, x):
        p1 = 1.0 + 0.5 * torch.tanh(self.p1_offset)
        p2 = 0.25 * torch.tanh(self.p2_offset)
        beta = 1.0 + 0.8 * torch.tanh(self.beta_offset)

        diff = (p1 - p2) * x
        return diff * torch.tanh(F.softplus(beta * diff)) + p2 * x
```

Table 7: Code-grounded summarization of the implemented child. The summarizer receives the task, proposed approach, and implemented code; its recorded output forms the child description supplied to the critic model.

Stage 3: solution summarization

System prompt

You are an expert ML solver. You are given a task, a candidate solution's code, and its short approach. Write a NEUTRAL, purely descriptive summary of what the solution does. Do NOT evaluate, rank, judge, compare, or predict scores.

User prompt

Task: mls__dl-activation-function

Shared input: insert the complete task description from Table 4.

Metric: **combined_score** (higher is better)

Approach

Shared input: insert the complete proposed approach from Table 5.

Code

Shared input: insert the complete implemented child code from Table 6.

Your task

Write a 120-200 word NEUTRAL descriptive summary of what the code does: task framing, model/algorithm, training/eval setup, and task-specific handling. PURELY DESCRIPTIVE — do NOT evaluate, rank, compare, or guess the score. Output only the summary.

Child summary

This solution implements a custom activation function for deep convolutional neural networks (ResNet, VGG, MobileNetV2) evaluated on CIFAR-10, CIFAR-100, and FashionMNIST. The approach, termed Mish-ACON-C, integrates the Activate Or Not (ACON) framework with the Mish activation function to create a parameterized, layer-specific non-linearity. The `CustomActivation` module defines three learnable scalar parameters per layer: `p1_offset`, `p2_offset`, and `beta_offset`, all initialized to zero. To maintain stability during training with the fixed SGD optimizer and restrict weight decay collapse, these offsets are passed through a `tanh` function to create bounded variables: a positive slope (`p1 = 1.0 + 0.5 * tanh(p1_offset)`), a negative slope (`p2 = 0.25 * tanh(p2_offset)`), and a transition sharpness (`beta = 1.0 + 0.8 * tanh(beta_offset)`).
During the forward pass, the module computes a scaled input `diff = (p1 - p2)* x` and applies the transformation `diff * tanh(softplus(beta * diff))+ p2 * x`. Because the offsets are zero-initialized, the function initially behaves exactly like the baseline Mish activation (`x * tanh(softplus(x))`), allowing the network to adjust its gating and asymptotes throughout the training process.

C.2 TASK UNDERSTANDING.

Table 8 presents the precomputed task context supplied to the critic model. It describes the evaluation objective, submission interface, datasets, input shapes, and preprocessing. This context is generated once per task and reused across candidates.

Table 8: Precomputed task understanding supplied to the critic model for activation-function design. This context is generated once per task and reused across candidates; it is shown here in its recorded wording.

Task-specific context for the critic model

Task understanding

Modality: Computer Vision (Image Classification)

Metric: Test accuracy (higher-is-better)

Overview

The task requires implementing a custom activation function as a PyTorch `nn.Module` to serve as a drop-in replacement for standard activations in ResNet, VGG, and MobileNetV2 architectures. The submission format is Python code modifying the `forward` method and optionally the `__init__` method of the `CustomActivation` class within an 18-line editable region in `pytorch-vision/custom_activation.py`. The implemented function must perform an element-wise or channel-wise operation that preserves the exact shape of the input tensor.

Data

* **Source Datasets:** CIFAR-10, CIFAR-100, and FashionMNIST.

* **Target Cardinality:** 10 classes for CIFAR-10 and FashionMNIST; 100 classes for CIFAR-100.

* **Input Image Dimensions:** All dataset images are resized to 32x32 pixels.

* **Channels:** CIFAR-10 and CIFAR-100 are natively 3-channel RGB. FashionMNIST is natively grayscale but is explicitly repeated across 3 channels (`x.repeat(3, 1, 1)`) in the data pipeline.

* **Activation Tensor Shapes:** The custom activation receives intermediate feature map tensors of varying shapes depending on the specific architecture (ResNet-20, ResNet-56, VGG-16, MobileNetV2) and layer depth.

* **Batch Size:** 128 for both training and testing dataloaders.

* **Normalization Statistics:**

* CIFAR-10: mean `(0.4914, 0.4822, 0.4465)`, std `(0.2023, 0.1994, 0.2010)`.

* CIFAR-100: mean `(0.5071, 0.4867, 0.4408)`, std `(0.2675, 0.2565, 0.2761)`.

* FashionMNIST: mean `(0.2860, 0.2860, 0.2860)`, std `(0.3530, 0.3530, 0.3530)`.

* **Data Augmentation:** Training sets apply `RandomCrop(32, padding=4)` and `RandomHorizontalFlip()`. Both train and test sets apply `ToTensor()` and normalization.

D Complete Prompt Instructions

Table 9 presents the complete input and output format used for critic training and inference.

E Soft-Rubric Reasoning Examples

Table 10 presents example critiques with freestyle and soft-rubric reasoning.

F Experiment Setup Details

F.1 Supervised Fine-Tuning Configuration

We train both critic model sizes for three epochs using Low-Rank Adaptation (LoRA) with rank 32. We use a learning rate of 1×10^{-5} for all SFT experiments and run for 3 epochs.

F.2 GRPO Configuration

For both 9B and 35B models, starting from the SFT checkpoints, we run GRPO for 300 rollout steps, with each step processing 256 idea inputs. For each idea, the critic generates a group of eight responses. We discard groups in which all responses receive the same reward, as these groups contain no relative learning signal for GRPO. We use the temperature of 0.7, learning rate of 1×10^{-6}, weight decay of 0.1, and KL coefficient of 0.01.

Table 9: Prompt template for the idea-level critic. The provided task-specific criteria must all be considered, while the critic may freely determine their relative importance and aggregation.

Component	Prompt content
Role	You are an expert ML solution critic. Given a parent solution and a candidate child idea, predict whether implementing the child idea will improve the task metric over the parent.
Decision stages	**Stage 1: Improvement.** Determine whether the child will `improve`, `lose`, or `tie` relative to the parent. **Stage 2: Win tier.** If the Stage-1 prediction is `improve`, determine whether the expected gain is `big`, `medium`, or `small`. Otherwise, output `NA`.
Task input	`{task_description}`
# Parent (baseline)	`{parent_idea_summary}`
# Child (candidate)	`{child_idea_summary}`
Task context	`{neutral_understanding_md}`
Task-specific criteria	`{task_specific_criteria_md}`
Soft-rubric reasoning	Assess the child idea against the parent solution using the provided task-specific criteria. 1. Evaluate every provided criterion. You may introduce an additional criterion only when it captures an important factor missing from the provided rubric. 2. For each criterion, state whether the child is `better`, `worse`, or `about the same` as the parent. 3. Determine the importance of each criterion for the current task, using either qualitative weights (`high`/`medium`/`low`) or approximate percentages, and briefly justify the weighting. 4. Explain how the weighted criterion-level comparisons are combined into the final prediction. The criteria constrain what must be evaluated, but the analysis, weighting, and aggregation strategy remain flexible.
Output format	`<criterion-level comparisons and explicit weighted aggregation>`
# Verdict	`[IMPROVEMENT] improve | lose | tie -- <one-sentence reason>` `[WINTIER] big | medium | small | NA -- <one-sentence reason>`
Output constraints	Use the exact tags `[IMPROVEMENT]` and `[WINTIER]`. Provide a calibrated and decisive prediction. The `[WINTIER]` prediction must be `NA` whenever the `[IMPROVEMENT]` prediction is `lose` or `tie`.

F.3 Policy GRPO Training Details

We initialize the idea-proposing policy from both Qwen-3.5-9B and Qwen-3.5-35B and train it on 95 MLE-Dojo and 46 MLS-Bench environments. For critic-guided reward assignment, we use the frozen Critic-9B and Critic-35B checkpoint obtained after SFT and GRPO for training idea proposer Qwen-3.5-9B and Qwen-3.5-35B respectfully. Each training step collects 16 parent-conditioned groups, with 16 candidate ideas per group. The initial parent for each task is its seed program. Whenever all 16 candidates in a group receive an IMPROVE verdict (i.e., an assigned reward of $+1$), we randomly select one child to become the task's new parent; otherwise, the parent remains unchanged. Each method receives a total budget of 10,000 ground-truth verifications across all training environments.

The policy generates only candidate ideas, while Gemini-3.1-Pro is responsible for implementing ideas and summarizing implementations. If a solution is selected to be verified, for non-timeout execution errors, Gemini-3.1-Pro is allowed up to three repair attempts. Both the initial implementation and subsequent repairs must strictly follow the proposed idea: Gemini-3.1-Pro is instructed to resolve

Table 10: Qualitative comparison between freestyle and soft rubric-based critic reasoning. The example is abridged for readability. Freestyle reasoning focuses primarily on the risk of overfitting, whereas soft rubric reasoning explicitly compares the parent and child under task-specific criteria before aggregating the judgments.

	Freestyle Reasoning	**Soft Rubric-Based Reasoning**
Analysis	The child proposes a custom U-Net trained from scratch on a subset of the available data, with images downsampled to 128×128. Although the pipeline is computationally feasible, training a CNN from scratch on a small medical-imaging dataset without pretrained weights or data augmentation is likely to cause severe overfitting. The parent instead relies on statistical priors, such as mean masks and spatial probabilities. These priors are robust on small datasets and can exploit the relatively consistent anatomical locations of the target organs. The freestyle critic therefore predicts that the child will generalize worse than the parent.	**1. Criteria considered.** *MRI handling and feature extraction:* whether the method captures spatial structure, intensity patterns, and patient-specific anatomical variation. *Dice and Hausdorff performance:* whether the method can produce accurate masks and boundaries under the task metric. ... **2. Criterion importance.** Feature extraction receives high importance (40%) because medical segmentation depends on learning spatial and intensity features. Metric alignment also receives high importance (30%), particularly because Hausdorff distance is sensitive to boundary errors and outlier pixels. ... **3. Parent–child comparison.** For feature extraction, the child is substantially better: the parent's statistical templates cannot capture patient-specific variation, whereas U-Net can learn nonlinear hierarchical features directly from the images. For metric alignment, the child is also better: U-Net's skip connections can produce more accurate and continuous segmentation boundaries than fixed statistical priors. .... **4. Aggregation.** After weighting the criterion-level comparisons, the expected benefits of learned spatial features and improved boundaries outweigh the child's increased risk of overfitting.
Verdict	`[IMPROVEMENT]` **lose** The child is likely to overfit because it trains a U-Net from scratch on only a subset of a small dataset. `[WINTIER]` **NA** The predicted outcome is not an improvement.	`[IMPROVEMENT]` **improve** The child replaces a rigid statistical prior with a model capable of learning task-specific spatial and intensity features. `[WINTIER]` **big** Moving from a non-learned statistical baseline to U-Net is expected to produce a substantial improvement in both mask accuracy and boundary quality.

implementation errors without introducing new algorithmic choices or modifying the substantive proposal.

For both 9B and 35B models, the critic-in-loop checkpoints are trained for around 110 steps while the verify-all checkpoints are trained for around 40 steps, targeting the 10,000 verification budget. We use learning rate of $1 \times 10^{-}6$ and weight decay of 0.1. The temperature used is 0.7.

G Additional Critic Evaluation Results

G.1 Full Freestyle and Soft-Rubric Ablation Results

Table 11 shows the results for critique using freestyle reasoning vs. soft rubric reasoning with all metrics.

G.2 Results across Sampled, Revised, and Guided Examples

Table 12 shows the detailed breakdown of results in Table 1. We show the results across the Sampled, Revised, and Guided subsets separately.

Table 11: Idea-critic performance on the 1.5K-example evaluation set. SFT substantially improves both model sizes, while criterion-guided GRPO provides consistent additional gains. Best results are shown in bold.

Model	Training	Acc	Dir-3	F1	5-Ladder
Qwen-3.5-9B	Base	49.9	41.3	58.3	22.4
Qwen-3.5-35B	Base	49.9	36.4	51.8	20.1
Qwen-3.5-9B	SFT, freestyle	61.0	54.6	71.5	31.9
Qwen-3.5-9B	SFT, soft-rubric	62.4	55.6	70.9	31.5
Qwen-3.5-9B	SFT+GRPO, freestyle	49.1	40.3	48.6	26.8
Qwen-3.5-9B	SFT+GRPO, soft-rubric	66.7	60.7	**75.6**	**41.5**

Table 12: Critic performance (%) across the Sampled, Revised, and Guided subsets. These subsets correspond to increasing levels of teacher assistance during critique collection. The best result in each column is shown in bold.

Model	Training	Improve Acc.			F1			Direction-3			5-Ladder		
		Sampled	Revised	Guided	Sampled	Revised	Guided	Sampled	Revised	Guided	Sampled	Revised	Guided
Qwen-3.5-9B	Base	54.2	50.6	45.0	54.2	59.2	54.5	46.8	42.6	34.6	34.4	20.0	12.0
Qwen-3.5-35B	Base	58.0	47.4	44.4	58.0	51.7	46.5	48.0	35.4	25.8	29.8	20.0	8.6
Gemini-3.1-Pro	Zero-shot	74.8	46.6	41.4	77.2	55.3	47.4	72.4	39.8	27.6	60.6	17.8	4.6
Qwen-3.5-9B	SFT	72.0	61.2	54.0	77.6	71.1	64.2	65.2	55.2	46.4	48.0	27.2	19.4
Qwen-3.5-9B	SFT+GRPO	72.4	65.4	61.4	77.5	74.2	72.7	69.0	63.6	52.8	56.0	37.8	28.0
Qwen-3.5-35B	SFT	75.0	61.6	60.6	80.1	74.9	67.1	69.0	60.2	51.6	49.8	37.4	28.2
Qwen-3.5-35B	SFT+GRPO	74.0	67.0	64.4	82.0	73.5	69.8	71.4	61.4	56.0	57.8	41.0	37.8

H OpenEvolve Evaluation Details

H.1 OpenEvolve Framework

Evolutionary search. OpenEvolve (Sharma, 2025) is an evolutionary coding framework for recursive self-improvement. It begins with an executable seed program and maintains a population of previously evaluated solutions. At each evolutionary round, OpenEvolve samples a parent solution, together with additional high-performing or diverse solutions that serve as inspirations, and prompts a language model to propose a modification. The resulting child program is executed by a task-specific evaluator, which returns empirical performance metrics and execution feedback. The evaluated child is then added to the population and can be selected as a parent or inspiration in subsequent rounds. Repeating this process allows improvements discovered in earlier rounds to be accumulated into progressively stronger solutions.

Population and island management. OpenEvolve combines a quality-diversity archive based on MAP-Elites with an island-based population structure. MAP-Elites partitions candidate solutions according to configurable behavioral or structural features and retains a high-performing solution within each occupied region, preserving diverse approaches rather than optimizing only a single lineage. Multiple islands maintain semi-independent populations and therefore explore different evolutionary trajectories. High-performing solutions are periodically migrated between islands, allowing useful discoveries from one population to influence the others while reducing premature convergence. Parent selection balances exploitation of high-performing solutions with exploration of diverse or less frequently sampled regions of the population. OpenEvolve also tracks the best verified solution separately so that it is preserved throughout evolution.

Configuration in our experiments. For each task and screening setting, we maintain three islands and evolve each island for 10 rounds. Exactly one selected idea is implemented and empirically verified in each island round, resulting in a total budget of 30 verification runs per task. We set migration interval as 4(migration happens every 4 evolution rounds). The idea proposer is using temperature of 0.8, and idea and implementation summarizer is using temperature of 0.4. All critic models are using temperature of 0.7. All settings begin from the same seed implementation and use

Table 13: The 30 tasks in MLS-Bench-Lite, grouped by their 12 official domains. Tasks marked with † were included in the task environments used for SFT data collection, while tasks marked with‡ were introduced only during critic model GRPO training. Unmarked tasks were held out from critic model training. We retain all 30 tasks to preserve the complete MLS-Bench-Lite evaluation suite.

Domain	#	Tasks
Language Models	3	`llm-dllm-demask-strategy`; `llm-pretrain-optimizer`; `llm-rl-importance-sampling`
Robotics	5	`jepa-planning`; `robo-diffusion-guidance`; `robo-diffusion-policy`; `robo-humanoid-sim2real-algo`; `robomimic-bc-loss`
Vision and Generation	3	`cv-3dgs-densification`; `cv-dbm-sampler`; `cv-vae-loss` †
Reinforcement Learning	1	`rl-value-discrete`
ML Systems and Efficient ML	3	`llm-ptq-algorithm`; `llm-qat-algorithm`; `mlsys-sparse-attention-inference`
AI for Science	3	`ai4bio-mutation-effect-prediction`; `ai4sci-inverse-diffusion-algo`; `ai4sci-pla-binding-affinity`
Optimization and Theory	2	`optimization-multi-objective`; `optimization-variance-reduction` ‡
Classical and Adaptive Learning	2	`ml-clustering-algorithm` †; `ml-dimensionality-reduction` †
Deep Learning	2	`cv-pooling-aggregation`; `dl-activation-function` †
Time Series and Forecasting	3	`quant-concept-drift`; `ts-exogenous-forecast` †; `ts-imputation` †
Structured and Causal Reasoning	2	`causal-discovery-discrete` ‡; `graph-generation` †
Trustworthy Learning	1	`security-membership-inference-defense` ‡
Total	**30**	**12 domains**

the same proposer, implementer, summarizer, population configuration, and verification budget; they differ only in whether and how candidate ideas are screened before execution.

H.2 Critic-Guided Candidate Selection

At each evolutionary round, Gemini-3.1-Pro generates a pool of 16 candidate ideas conditioned on the current parent solution. For each candidate, the critic generates 16 critiques comparing it with the parent and predicts one of five outcome labels. We rank candidates by their predicted labels in the order BigWin > MediumWin > SmallWin > Tie > Lose and calculate an average rank for each candidate idea. Only the candidate idea with the highest average rank for execution. Consequently, exactly one of the 16 candidates consumes a real verification run, while the remaining candidates are filtered without executing their training pipelines.

We apply the same candidate-generation and selection procedure when using Gemini-3.1-Pro, the base Qwen-3.5-9B/35B models, or our trained Critic-9B/35B as the selector. Vanilla OpenEvolve does not apply this critic-based screening stage.

H.3 Task Descriptions

MLS-Bench-Lite. MLS-Bench (Lyu et al., 2026) evaluates whether an ML agent can improve a targeted component of an existing machine learning system. Each task provides a fixed research scaffold, relevant source code, baseline implementations, a constrained edit surface, and an executable evaluator. The agent must propose and implement an algorithmic modification that improves performance across the task's prescribed evaluation settings. These tasks therefore test not only whether an agent can produce executable code, but also whether it can identify ML modifications that yield robust empirical improvements. We use MLS-Bench-Lite, the official 30-task subset of the full 140-task benchmark. The subset preserves coverage of all 12 MLS-Bench domains: language models, robotics, vision and generation, reinforcement learning, ML systems and efficient ML, AI for science, optimization and theory, classical and adaptive learning, deep learning, time series and forecasting, structured and causal reasoning, and trustworthy learning. The 30 tasks cover a broad range of ML development problems, including model architecture design, training objectives, optimization algorithms, inference procedures, data processing, and scientific modeling. The complete task list and domain assignments are provided in Table 13.

Table 14: Student and outcome-conditioned teacher critiques on an LLM PTQ example. Idea descriptions are summarized; critique excerpts are quoted verbatim, with emphasis added. The critiques were regenerated from an archived parent–child pair for qualitative inspection.

Task: LLM post-training quantization (INT4; WikiText-2 perplexity, lower is better).
Parent idea: Apply GPTQ with Hessian-based error compensation, dynamically computed group quantization parameters, and damped Cholesky decomposition.
Child idea: Integrate activation-aware channel scaling into GPTQ, use asymmetric group quantization, and add adaptive dampening while retaining GPTQ error compensation.
Empirical outcome: Perplexity decreases from 5.0428 to 4.9906 (1.04% relative reduction); label: IMPROVE.

Student: without outcome hint	**Teacher: with outcome hint**
Trained critic with the SDFT LoRA adapter enabled.	Frozen critic; receives the measured parent/child metrics and outcome label.
Quantization reasoning	**Quantization reasoning**
"While the CHILD introduces Asymmetric Quantization (AQ) and AWQ scaling, AQ requires a 1-bit zero point that can cause overflow when dequantized back to FP16. This negates the theoretical benefit of using finer step sizes, and the resulting perplexity remains statistically indistinguishable from the PARENT's symmetric GPTQ method."	"By incorporating AWQ scaling into GPTQ, the CHILD effectively reduces the quantization error variance on salient (high-activation) channels while **preserving the error-compensation mechanism of GPTQ**. This leads to a lower perplexity (4.9906 vs 5.0428)."
Interpretation of the proposed modification	**Interpretation of the proposed modification**
"The PARENT uses the full Hessian matrix for error propagation, whereas the CHILD only uses the diagonal (the exact same information utilized by AWQ). The off-diagonal terms provide valuable error compensation that the **CHILD discards**, offsetting its scaling advantage."	"The CHILD introduces a well-established and highly effective improvement over the PARENT by **combining AWQ's salient channel protection with GPTQ's error compensation**."
Predicted outcome: TIE (incorrect).	**Predicted outcome:** IMPROVE, with a small win tier (outcome correct).

Scoring. Because the tasks use heterogeneous evaluation metrics, MLS-Bench defines a task-specific scoring specification for converting raw evaluation results into normalized task scores. These specifications account for metric direction, task-specific baseline anchors, theoretical or practical performance bounds, and, when applicable, multiple evaluation settings. We use the official scoring specification for each task without modification. We aggregate the normalized task scores within each domain using the arithmetic mean and report the resulting domain-level performance in Table 2.

Overlap with critic model training environments. A subset of the MLS-Bench-Lite tasks also appears among the task environments used to collect training data for our critic models. We retain these tasks rather than removing them because our goal is to evaluate the complete official MLS-Bench-Lite suite and preserve its intended cross-domain composition.

H.4 OPSD TEACHER VS STUDENT CRITIQUE REASONING TRAJECTORY

Table 14 illustrates how outcome conditioning can shift a critique from dismissing a proposed modification to identifying a mechanism consistent with the observed improvement. This qualitative comparison does not independently validate the teacher's causal explanation.

Table 15: Policy GRPO under a fixed budget of 10,000 empirical verifications. We compare the base policy, verify-all GRPO, and critic-model-in-the-loop GRPO for Qwen-3.5-9B and Qwen-3.5-35B. Scores are averaged within each of the 11 retained MLS-Lite-Bench domains, and the best result in each row is shown in **bold**.

Domain	Qwen-3.5-9B			Qwen-3.5-35B		
	Base	Verify-all	Critic-in-loop	Base	Verify-all	Critic-in-loop
Language Models	9.24	12.17	12.04	11.94	**12.90**	12.62
Robotics	7.91	18.83	15.87	16.83	21.20	**23.46**
Vision and Generation	11.66	6.61	13.23	10.83	11.03	**14.07**
ML Systems and Efficient ML	21.88	23.99	**36.16**	11.39	16.87	30.99
AI for Science	12.49	**14.17**	13.86	11.09	8.34	12.61
Optimization and Theory	18.56	20.03	18.08	17.75	9.76	**25.13**
Classical and Adaptive Learning	16.95	19.80	22.85	28.90	**30.23**	29.88
Time Series and Forecasting	9.21	11.93	**30.97**	9.88	14.85	13.93
Structured and Causal Reasoning	34.48	23.60	32.40	19.98	18.52	**36.32**
Deep Learning	41.30	33.05	**48.59**	40.37	41.90	44.50
Trustworthy Learning	22.60	31.10	**33.23**	26.40	26.65	26.50
Domain Average	18.75	19.57	**25.21**	18.67	19.29	24.55

I More details on Critic-in-loop RL

I.1 Results for Policy Model GRPO on MLS-Lite

I.2 Ablation on the Critic Model as Reward Model

We investigate whether the trained critic can support policy optimization without empirical verification during training. Starting from the base Qwen-3.5-35B policy, we perform GRPO using rewards provided entirely by the frozen Critic-35B, with no fallback to the ground-truth verifier. For each candidate idea, we aggregate k critic rollouts by majority voting over IMPROVE, TIE, and LOSE, assigning a reward of $+1$ to IMPROVE and -1 otherwise. We conduct two sweeps: varying the number of policy training steps from 0 to 160, and varying $k \in \{1, 4, 8, 16, 32\}$ while fixing training to 40 steps.

Figure 5 shows that policy performance generally improves as the number of training steps increases on both `ML Clustering Algorithm task` and `Graph Generation` task. The trend is particularly clear for `Graph Generation`, indicating that rewards from critic model continue to provide a useful learning signal even without access to empirical verification during training. Increasing the number of critic model rollouts k also improves performance on `AI4Bio Mutation Effect Prediction` task and `Graph Generation` task, but the gains are comparatively modest and begin to saturate. In particular, increasing k beyond 16 provides limited additional benefit while proportionally increasing inference computational cost. We therefore use $k = 16$ as a practical trade-off between reward reliability and critic model inference computational cost.

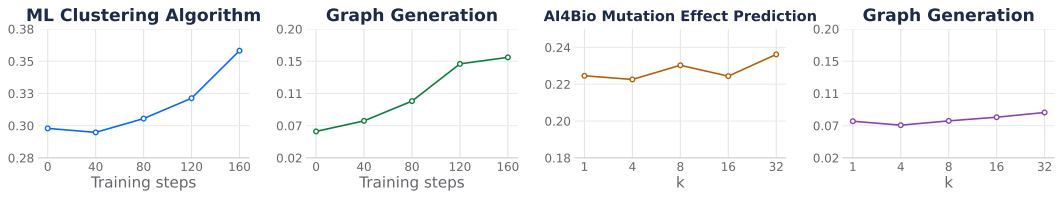


Figure 5: Critic-model-only reward ablation. The left panels vary the number of policy-training steps, while the right panels vary the number of critic model rollouts k with training fixed to 40 steps.