Learning to Ideate for Machine Learning Engineering Agents

Yunxiang Zhang[1,2*] **Kang Zhou**[1†] **Zhichao Xu**[1] **Kiran Ramnath**[1]
Yun Zhou[1] **Sangmin Woo**[1] **Haibo Ding**[1†] **Lin Lee Cheong**[1]
[1]AWS AI Labs [2]University of Michigan
{yxzyx,zhoukang,xzhichao,raxkiran,yunzzhou,
sangminw,hbding,lcheong}@amazon.com

Abstract

Existing machine learning engineering (MLE) agents struggle to iteratively optimize their implemented algorithms for effectiveness. To address this, we introduce **MLE-IDEATOR**, a dual-agent framework that separates ideation from implementation. In our system, an implementation agent can request strategic help from a dedicated IDEATOR. We show this approach is effective in two ways. First, in a training-free setup, our framework significantly outperforms implementation-only agent baselines on MLE-Bench. Second, we demonstrate that the IDEATOR can be trained with reinforcement learning (RL) to generate more effective ideas. With only 1K training samples from 10 MLE tasks, our RL-trained Qwen3-8B IDEATOR achieves an 11.5% relative improvement compared to its untrained counterpart and surpasses Claude Sonnet 3.5. These results highlights a promising path toward training strategic AI systems for scientific discovery.

1 Introduction

Artificial intelligence (AI) agents capable of building machine learning models offer great potential to accelerate scientific discovery (Swanson et al., 2024; Lu et al., 2024; Chan et al., 2025). While AI agents powered by large language models (LLMs) have made significant progress in relevant fields such as software development (Wang et al., 2025b; Jimenez et al., 2024; Yang et al., 2024), their typical designs are insufficient for autonomous machine learning engineering (MLE). They are often limited to implementing a single valid solution, ceasing exploration prematurely rather than continuously optimizing algorithms to enhance performance. In MLE tasks, a merely valid solution is insufficient, as the task demands strategic iteration—testing alternative models and hyperparameters, refining data and features to achieve state-of-the-art results (Chan et al., 2025; Qiang et al., 2025; Zhang et al., 2025b).

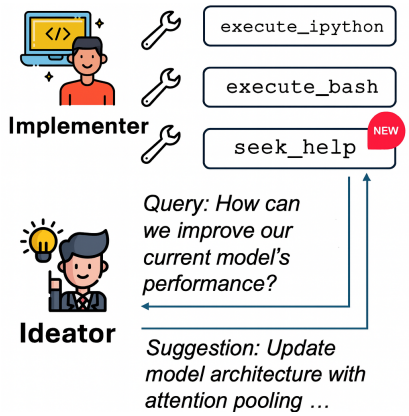

Figure 1: Overview of **MLE-IDEATOR**. The IMPLEMENTER uses a <seek_help> action to solicit strategic guidance from a dedicated IDEATOR. This separation improves performance over single-agent baselines and enables the IDEATOR to be optimized via reinforcement learning with execution-based rewards.

While recent work has sought to develop agents capable of strategic iteration, typically by training small language models (SLMs) as MLE agents via reinforcement learning (RL) (Liu et al., 2025b; Yang et al., 2025b), these approaches typically rely on a single-agent paradigm that conflates strategic reasoning (idea proposal) with coding proficiency (implementation). This conflation imposes a critical trade-off: small models lack the coding capacity to implement complex solutions, yet training them to master both skills requires inefficient multi-turn rollouts with sparse rewards. To address this structural limitation, we ask: *Can decoupling ideation from implementation enable small models to efficiently learn strategies that guide stronger models?*

We answer this question by designing a novel dual-agent framework **MLE-IDEATOR** (Figure 1) that explicitly separates ideation from implementation. When the Implementer's progress plateaus,

[*]Work done during an internship at Amazon.
[†]Corresponding authors.

Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics
Volume 2: Short Papers, pages 436–447
March 24-29, 2026

it can invoke a new <seek_help> action to solicit high-level strategies from the IDEATOR. This design facilitates *superalignment* (Burns et al., 2024), where a lightweight IDEATOR model serves as a strategic guider for a more capable Implementer, directing algorithm refinements without needing to retrain the Implementer itself. A well-trained IDEATOR can thus be reused as a plug-in to augment the performance of various implementer models.

We first demonstrate on the MLE-Bench (Chan et al., 2025) that training-free ideation improves performance over implementation-only agent baselines. To further enhance ideation capability, we introduce an RL pipeline to train the IDEATOR. Each proposed idea is executed by a frozen Implementer in a *single* step and rewarded if it improves performance. The IDEATOR is then updated using the GRPO (Shao et al., 2024) algorithm to maximize this execution-based reward, thereby efficiently avoiding costly multi-step rollouts. Our experiments show that the Qwen3-8B model, after RL training on just 10 MLE tasks, not only outperforms its untrained counterpart but also surpasses the powerful Claude Sonnet 3.5 as an IDEATOR. These results highlight a promising path toward teaching LLM agents to generate high-impact ideas for scientific discovery.

Our contributions are threefold:

- We introduce MLE-IDEATOR, a novel dual-agent framework that separates high-level ideation from low-level implementation for MLE tasks.
- We design an efficient RL pipeline to train an IDEATOR with execution-based rewards, minimizing reliance on costly multi-step rollouts.
- We demonstrate strong performance gains on MLE-Bench, showing that an RL-trained small Ideator can outperform more powerful LLMs in guiding the Implementer.

2 Related Work

Single-Agent Frameworks for MLE. Coding agents such as CodeAct (Wang et al., 2025b), MLAB (Huang et al., 2024), and RepoMaster (Wang et al., 2025a) execute MLE tasks using basic tools like Python and Bash but lack dedicated ideation, often stopping at the first valid solution. Parallel exploration frameworks like AIDE (Jiang et al., 2025b), AIRA (Toledo et al., 2025), and ML-Master (Liu et al., 2025a) extend this via tree search across multiple solution branches. However, these methods rely on expensive code-space search and static prompting, limiting exploration to implementation rather than high-level ideation. In contrast, we decouple ideation from coding, enabling efficient idea-space exploration without costly tree-search orchestration.

Multi-Agent Frameworks for MLE. Multi-agent frameworks like MLZero (Fang et al., 2025), InternAgent (Zhang et al., 2025a), Agent Laboratory (Schmidgall et al., 2025), and R&D-Agent (Yang et al., 2025c) decompose tasks using specialized roles (e.g., planner, experimenter). Yet, their coordination relies on complex, rigid, and prompt-driven workflows. Our work differs in two ways. First, we employ a dynamic workflow where the Implementer solicits on-demand strategic guidance via a <seek_help> action, ensuring collaboration is context-driven rather than predefined. Second, unlike prompt-only systems, we apply reinforcement learning directly to the ideation model, optimizing it to generate effective algorithmic suggestions based on performance outcomes.

RL-Trained MLE Agents. Reinforcement learning has recently been applied to MLE agents (Liu et al., 2025b; Yang et al., 2025b), but these works focus on training single-agent implementers for code generation, neglecting the strategic challenge of ideation. By contrast, we train the IDEATOR itself with RL to directly optimize idea quality via performance feedback. This lightweight mechanism enables the IDEATOR to learn high-level refinements.

3 Methods

3.1 MLE-IDEATOR

To decouple ideation from implementation in machine learning engineering tasks, we propose a dual-agent framework in Figure 1 that consists of two primary components: a Implementer agent responsible for implementation and an IDEATOR agent for providing high-level suggestions. We instantiate the Implementer with the CodeAct agent framework (Wang et al., 2025b), which provides an action space for executing Python and Bash commands (<execute_ipython> and <execute_bash>). We introduce a new action, <seek_help>, which the Implementer can invoke as needed when it experiences plateaued perfor-

mance, or identifies a potential optimization opportunity. The query is formatted as follows:

Implementer Agent Query via <seek_help>

```
<seek_help>
PROBLEM_STATEMENT:
<one sentence describing the current blocking issue>

ATTEMPTS_SO_FAR:
<a short bullet list of what you already tried>

GOAL:
<one sentence on what success looks like for this step>
</seek_help>
```

The IDEATOR receives this query along with the full trajectory of the Implementer's work (including code, logs, performance) and provides a structured suggestion:

IDEATOR Agent Response for <seek_help>

```
ANALYSIS_ON_CURRENT_PROGRESS:
<Briefly state whether to keep refining the present
approach or revert to a prior solution and pursue a
new path.>

ACTION:
<One imperative command or code block the agent must
execute next.>

RATIONALE:
<Concise justification for why this action is optimal
right now.>
```

This structure makes the suggestions easily comprehendable and actionable for the Implementer. The Implementer receives the IDEATOR's output as an observation and integrates the suggested action into its workflow in the subsequent step. This protocol enables targeted, context-aware collaboration.

3.2 MLE-IDEATOR-RL

To enable the IDEATOR to learn from experience, we introduce a reinforcement learning framework to train the IDEATOR to propose more effective ideas.

State and Action Space. At each timestep t where the Implementer issues a `<seek_help>` action, the IDEATOR receives a comprehensive snapshot of the Implementer agent's progress as the current state representation $s_t = (\mathcal{D}, \tau_{1:t}, p_t, C_t)$, consisting of the task description $\mathcal{D}$, the current trajectory $\tau_{1:t} = (a_1, o_1, ..., a_t)$, which includes all previous actions and observations, the current ML solution's performance score p_t, and the full ML solution code C_t. The IDEATOR's action is to generate a natural language suggestion α with a structured format: $\alpha = (\text{ANALYSIS}, \text{ACTION}, \text{RATIONALE})$.

Reward Function. To directly optimize for effective ideas, we define a discrete, three-level reward function $R_t(\alpha)$ that evaluates the outcome of applying an idea α:

$$R_t(\alpha) = \begin{cases} +1 & \text{if } p_{t+1}(\alpha) > p_t \\ 0 & \text{if } p_{t+1}(\alpha) \leq p_t \text{ or execution fails} \\ -1 & \text{if format errors,} \end{cases} \tag{1}$$

where p_t is the solution performance before the idea, and $p_{t+1}(\alpha)$ is the new performance after the Implementer executes revised code based on the idea α in a *single* step. The reward function incentivizes the IDEATOR to propose ideas that improve performance and penalizes those that are ineffective or invalid.

RL Training Pipeline. To avoid regenerating agent trajectories online at each training step, we first run MLE-IDEATOR offline to collect a pool of states whose trajectories end with the `<seek_help>` action. These states are then used to construct prompts for the IDEATOR (see Appendix D). We employ GRPO (Shao et al., 2024) as our RL algorithm for its efficiency and stability. The training objective is to maximize the expected reward for the ideas generated by the IDEATOR (Appendix A.1). Each training step begins with an **ideation (rollout) phase**, where we sample G candidate ideas $\{\alpha_1, ..., \alpha_G\}$ from the IDEATOR's current policy. Next, during the **execution (reward calculation) phase**, a frozen Implementer agent attempts to implement each valid idea in a single step.[1] If execution succeeds, we evaluate the new ML solution to compute a performance-based reward defined in Equation 1. Finally, in the **backpropagation phase**, we update the IDEATOR's policy parameters using the GRPO loss function (Appendix A.2). This formulation enables direct optimization of idea effectiveness through performance-based feedback.

4 Experiments

4.1 Experimental Setup

Dataset and Baselines. Our experiments are conducted on the **MLE-Bench** dataset (Chan et al., 2025). For our training tasks, we select 10 diverse Kaggle tasks (Appendix B.1) that span multiple data modalities and whose solutions can typi-

[1]We restrict execution to a single step for simplicity and efficiency, though more turns could reduce execution failure through agent self-debugging.

System	IDEATOR	Avg@3	Best@3
Implementation-only Agent Baselines			
CodeAct + GPT-4o	-	47.9	51.7
AIDE + GPT-4o	-	49.6	50.7
AIDE + Sonnet 3.5	-	50.7	53.8
Implementer = Claude Sonnet 3.5			
CodeAct	-	50.6	52.8
MLE-IDEATOR	Sonnet 3.5	**58.5**	60.9
	Qwen3-8B	53.2	56.6
MLE-IDEATOR-RL	Qwen3-8B-RL	58.4	**63.1**
Implementer = Qwen3-8B			
CodeAct	-	25.4	25.9
MLE-IDEATOR	Sonnet 3.5	28.0	28.3
	Qwen3-8B	25.2	25.6
MLE-IDEATOR-RL	Qwen3-8B-RL	**29.8**	**30.1**

Table 1: Main results on 51 held-out MLE-Bench tasks.

IDEATOR	Avg@3	Best@3
Implementer = Claude Sonnet 3.5		
-	69.7	72.0
NULL IDEA	68.7	72.8
VAGUE IDEA	75.0	76.3
Claude Sonnet 3.5	**80.1**	**83.8**

Table 2: Impact of idea quality on MLE-IDEATOR performance under 22 low-complexity MLE-Bench tasks. NULL IDEA and VAGUE IDEA denote non-LLM ablations where the IDEATOR is replaced with a fixed template response. NULL IDEA always returns "*I have no suggestions for improving the solution. Please proceed using your best judgment.*" and VAGUE IDEA always returns "*Keep improving the performance of your solution.*"

cally be executed within 20 minutes, a criterion that enables faster reward calculation. To collect training data from these 10 tasks, we run the MLE-IDEATOR framework 90 times for each using Claude Sonnet 3.5 as the backbone model and randomly sample 1000 states to serve as training prompts (100 per task) and 100 states for validation (10 per task).[2] Our evaluation is performed on a held-out set of 51 tasks (21 low-complexity and 30 medium-complexity), excluding the 15 high-complexity tasks where agents consistently struggle to produce a valid submission within the time limit. We test several configurations using Qwen3-8B (Yang et al., 2025a) and Claude Sonnet 3.5 as the backbone LLMs for the IDEATOR and Implementer roles. We compare three configurations: a **CodeAct** baseline where the Implementer agent solves the task alone; an **MLE-IDEATOR (Prompting)** setup where the Implementer is paired with a prompted IDEATOR and can use the `<seek_help>` action; and our **MLE-IDEATOR-RL (RL-Trained)** approach, which replaces the prompted Qwen3-8B IDEATOR with our RL-trained version. During evaluation, all agents are allotted a maximum of 50 steps and a one-hour runtime per task. We show the hyperparameters of RL training in Appendix B.2.

Evaluation Metric. We evaluate agent performance using **Avg@3** and **Best@3** normalized scores (Jiang et al., 2025a), averaged across all tasks. For each task, an agent is run three independent times. Since the native evaluation metrics (e.g., accuracy, cross-entropy loss) vary in scale from one Kaggle task to another, we normalize the raw score of each run against human performance on the Kaggle leaderboard to a common 0-100 scale: $\max(0, 100 \times \frac{\text{agent\_score} - \text{worst\_human\_score}}{\text{best\_human\_score} - \text{worst\_human\_score}})$. The Best@3 is the maximum of these three normalized scores, while the Avg@3 is their average, excluding any runs that failed to produce a valid submission. An agent receives a score of zero for a task only if all three runs fail to produce any valid solutions.

4.2 Results and Analysis

Prompted and Reinforced Ideation Boost Performance. Table 1 shows that prompted ideation consistently boosts performance over implementation-only agent baselines. With Claude Sonnet 3.5 as the Implementer, pairing it with a prompted IDEATOR yields strong improvements: Avg@3 rises from 50.6 (CodeAct baseline) to 58.5 with a Sonnet IDEATOR and 53.2 with a Qwen3-8B IDEATOR. Reinforcement learning further enhances effectiveness, as the Qwen3-8B-RL IDEATOR achieves a Best@3 of 63.1, surpassing even the powerful Sonnet IDEATOR. Even with the weaker Qwen3-8B Implementer, ideation proves valuable: pairing with Sonnet 3.5 lifts performance from 25.4 to 28.0, and the RL-trained Qwen3-8B IDEATOR achieves a stronger result at 29.8. These results show that *prompted ideation consistently improves implementers, and RL enables smaller* IDEATORS *to surpass stronger prompted ones.*

[2]Each run may trigger multiple `<seek_help>` calls, producing over 1,000 states ending with that action. We sample the required training and validation sets from this pool.

Idea Type	Qwen3-8B	Qwen3-8B-RL	Δ
Data Preparation	13.4	16.5	+3.1
Feature Engineering	13.4	20.9	+7.5
Model Architecture	28.5	25.3	-3.2
Model Training	32.4	27.5	-4.9
Hyperparameter Tuning	7.3	6.6	-0.7
Others	5.0	3.3	-1.7

Table 3: Comparison of idea type distributions between Qwen3-8B and Qwen3-8B-RL. Values indicate the proportion (%) of ideas in each category, with Δ showing changes after RL training (cells shaded green for increases and red for decreases).

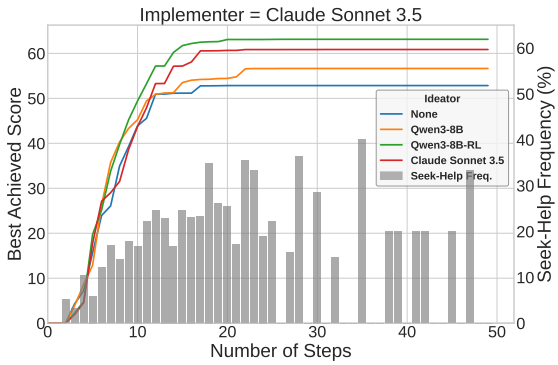

Figure 2: For each IDEATOR model, we show the agent's best achieved normalized score so far in a trajectory averaged over all tasks w.r.t the number of steps in trajectory. We also plot the frequency of <seek_help> actions at each steps, aggregated over all IDEATORS.

Idea Quality Matters. Is the performance boost driven by the quality of ideas, or simply by adding the <seek_help> action and receiving some form of feedback, even if it is uninformative? Table 2 ablates this factor by replacing the LLM-based IDEATOR with fixed template outputs. Using a NULL IDEA yields performance comparable to having no IDEATOR, since the Implementer receives no meaningful guidance beyond its own trajectory. A VAGUE IDEA, which always provides a generic encouragement, produces a slight improvement, likely because it prompts the Implementer to continue refining rather than stopping prematurely. However, this effect is small compared to the substantial gains achieved with specific, contextual ideas from the Claude Sonnet 3.5 IDEATOR. These results confirm that *high-quality, actionable guidance, rather than reflection or additional interaction, drives the performance improvement of* MLE-IDEATOR.

RL Aligns Idea Generation with Empirically Effective Strategies. Not all ideas contribute equally to performance, so we analyze which types are most effective. We define an idea as *effective* if it improves the ML solution's performance. To this end, we prompt a strong LLM (Claude Sonnet 4) to classify each idea into six categories: Data Preparation, Feature Engineering, Model Architecture, Model Training, Hyperparameter Tuning, and Others (full prompt in Appendix E). As shown in Figure 4 (Appendix C), **Feature Engineering** and **Data Preparation** ideas are typically more effective, whereas **Model Training** and **Hyperparameter Tuning** are less reliable. Building on this observation, Table 3 shows that *RL training shifts the idea distribution toward empirically effective categories* and reduces the frequency of less effective ones, thereby improving overall idea quality. We show concrete examples of the IDEATOR's successful and unsuccessful suggestions in Appendix F.

Ideation Drives Mid-Trajectory Refinement. Figure 2 correlates agent performance with <seek_help> behavior. Initially (<12 steps), the Implementer focuses on establishing a baseline with minimal help-seeking. Once an initial submission is made, <seek_help> frequency spikes. This IDEATOR-guided exploration drives immediate performance gains, allowing the system to escape local optima where single-agent baselines stagnate. Performance generally plateaus around step 20 due to diminishing returns on refinements and the one-hour execution limit. Future integration with inference-time scaling (Snell et al., 2024; Zhu et al., 2025; Shen et al., 2025) could unlock further gains via Best-of-N idea sampling.

5 Conclusion

We introduce MLE-IDEATOR, a dual-agent framework that decouples strategic ideation from implementation for machine learning engineering tasks. Our experiments on MLE-Bench demonstrate that this approach significantly outperforms implementation-only baselines. Furthermore, by training the Ideator with reinforcement learning using execution-based rewards, our Qwen3-8B model achieves an 11.5% relative improvement over its prompted counterpart and surpasses the proprietary Claude Sonnet 3.5, successfully aligning ideation with high-impact strategies. Our work provides a promising training recipe for LLM agents that can automate AI research.

Limitations

Additional Inference Cost. The MLE-IDEATOR framework incurs a higher inference cost than the implementation-only agent baseline, i.e., CodeAct (Wang et al., 2025b). When using Claude Sonnet 3.5 for both the Implementer and IDEATOR roles, a single run of our framework costs 1.4 USD on average, compared to 0.9 USD for the baseline. This increased cost stems from two primary factors:

- **Additional Steps:** The dual-agent system averages more steps per run (17.4 vs. 15.0) due to the introduction of the <seek_help> action required to query the IDEATOR.
- **Expanded Context:** To provide the IDEATOR with sufficient history, its prompt includes the entire agent trajectory (truncated to 32,000 tokens). This significantly increases the number of input tokens per call.

We expect that this cost can be mitigated. Future work could focus on developing summarization or pruning techniques (Wang et al., 2025a; Mei et al., 2025) to reduce the context length of the trajectory without sacrificing the quality of the generated ideas.

Additionally, the inference latency introduced by ideation is minor relative to the end-to-end ML pipeline, as each <seek_help> action is just a lightweight LLM call compared to the far more time-consuming code execution, model training, and evaluation steps of each ML task.

Resource-Intensive RL Training. The reinforcement learning process is resource-intensive, as calculating the reward for each proposed idea requires executing a full ML solution, which often involves training a neural network on a GPU. In our training setup, the reward calculation is distributed across 16 nodes, each with 8 A10G (24GB) GPUs, where each GPU is dedicated to running the solution for a single candidate idea (Appendix B.2). Our use of 128 A10 GPUs was a choice to accelerate reward computation with parallel ML solution execution, but not a strict requirement. The same procedure can be run with fewer GPUs by executing candidate solutions sequentially, trading off training time for compute resources.

A primary bottleneck to scalability lies in the requirement of a full execution to evaluate each generated idea. To mitigate this limitation, future work could focus on developing a proxy reward model. Such a model, by predicting an idea's effectiveness without costly execution (Park et al., 2025; Anugraha et al., 2025; Wen et al., 2025), would significantly enhance the scalability of the training process.

References

David Anugraha, Genta Indra Winata, Chenyue Li, Patrick Amadeus Irawan, and En-Shiun Annie Lee. 2025. Proxylm: Predicting language model performance on multilingual tasks via proxy models. In *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 1981–2011. Association for Computational Linguistics.

Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, Ilya Sutskever, and Jeffrey Wu. 2024. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

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

Haoyang Fang, Boran Han, Nick Erickson, Xiyuan Zhang, Su Zhou, Anirudh Dagar, Jiani Zhang, Ali Caner Türkmen, Cuixiong Hu, Huzefa Rangwala, Ying Nian Wu, Yuyang Wang, and George Karypis. 2025. Mlzero: A multi-agent system for end-to-end machine learning automation. *CoRR*, abs/2505.13941.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. Mlagentbench: Evaluating language agents on machine learning experimentation. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Minqi Jiang, Andrei Lupu, and Yoram Bachrach. 2025a. Bootstrapping task spaces for self-improvement. *arXiv preprint arXiv:2509.04575*.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang

Wu. 2025b. Aide: Ai-driven exploration in the space of code. *ArXiv*, abs/2502.13138.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Weinan E, and Siheng Chen. 2025a. Ml-master: Towards ai-for-ai via integration of exploration and reasoning. *CoRR*, abs/2506.16499.

Zexi Liu, Jingyi Chai, Xinyu Zhu, Shuo Tang, Rui Ye, Bolun Zhang, Lei Bai, and Siheng Chen. 2025b. Ml-agent: Reinforcing LLM agents for autonomous machine learning engineering. *CoRR*, abs/2505.23723.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob N. Foerster, Jeff Clune, and David Ha. 2024. The AI scientist: Towards fully automated open-ended scientific discovery. *CoRR*, abs/2408.06292.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. 2025. A survey of context engineering for large language models. *CoRR*, abs/2507.13334.

Jungsoo Park, Ethan Mendes, Gabriel Stanovsky, and Alan Ritter. 2025. Look before you leap: Estimating llm benchmark scores from descriptions. *Preprint*, arXiv:2509.20645.

Rushi Qiang, Yuchen Zhuang, Yinghao Li, Dingu Sagar V. K, Rongzhi Zhang, Changhao Li, Ian Shu-Hei Wong, Sherry Yang, Percy Liang, Chao Zhang, and Bo Dai. 2025. Mle-dojo: Interactive environments for empowering LLM agents in machine learning engineering. *CoRR*, abs/2505.07782.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Zicheng Liu, and Emad Barsoum. 2025. Agent laboratory: Using LLM agents as research assistants. *CoRR*, abs/2501.04227.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300.

Junhong Shen, Hao Bai, Lunjun Zhang, Yifei Zhou, Amrith Setlur, Shengbang Tong, Diego Caples, Nan Jiang, Tong Zhang, Ameet Talwalkar, and Aviral Kumar. 2025. Thinking vs. doing: Agents that reason by scaling test-time interaction. *CoRR*, abs/2506.07976.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pages 1279–1297. ACM.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *CoRR*, abs/2408.03314.

Kyle Swanson, Wesley Wu, Nash L. Bulaong, John E. Pak, and James Zou. 2024. The virtual lab: Ai agents design new sars-cov-2 nanobodies with experimental validation. *bioRxiv*.

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Mario Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Maria Lupidi, Andrei Lupu, Roberta Raileanu, Kelvin Niu, Tatiana Shavrina, Jean-Christophe Gagnon-Audet, Michael Shvartsman, Shagun Sodhani, Alexander H. Miller, Abhishek Charnalia, and 6 others. 2025. AI research agents for machine learning: Search, exploration, and generalization in mle-bench. *CoRR*, abs/2507.02554.

Huacan Wang, Ziyi Ni, Shuo Zhang, Shuo Lu, Sen Hu, Ziyang He, Chen Hu, Jiaye Lin, Yifu Guo, Yuntao Du, and Pin Lyu. 2025a. Repomaster: Autonomous exploration and understanding of github repositories for complex task solving. *CoRR*, abs/2505.21577.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 2 others. 2025b. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Jiaxin Wen, Chenglei Si, Yueh-han Chen, He He, and Shi Feng. 2025. Predicting empirical AI research outcomes with language models. *CoRR*, abs/2506.00794.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 40 others. 2025a. Qwen3 technical report. *CoRR*, abs/2505.09388.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*

38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024.

Sherry Yang, Joy He-Yueya, and Percy Liang. 2025b. Reinforcement learning for machine learning engineering agents. *arXiv preprint arXiv:2509.01684*.

Xu Yang, Xiao Yang, Shikai Fang, Bowen Xian, Yuante Li, Jian Wang, Minrui Xu, Haoran Pan, Xinpeng Hong, Weiqing Liu, Yelong Shen, Weizhu Chen, and Jiang Bian. 2025c. R&d-agent: Automating data-driven AI solution building through llm-powered automated research, development, and evolution. *CoRR*, abs/2505.14738.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, and 16 others. 2025. DAPO: an open-source LLM reinforcement learning system at scale. *CoRR*, abs/2503.14476.

Bo Zhang, Shi Feng, Xiangchao Yan, Jiakang Yuan, Zhiyin Yu, Xiaohan He, Songtao Huang, Shaowei Hou, Zheng Nie, Zhilong Wang, Jinyao Liu, Runmin Ma, Tianshuo Peng, Peng Ye, Dongzhan Zhou, Shufei Zhang, Xiaosong Wang, Yilan Zhang, Meng Li, and 5 others. 2025a. Novelseek: When agent becomes the scientist - building closed-loop system from hypothesis to verification. *ArXiv*, abs/2505.16938.

Yunxiang Zhang, Muhammad Khalifa, Shitanshu Bhushan, Grant D. Murphy, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. 2025b. MLRC-Bench: Can language agents solve machine learning research challenges? *CoRR*, abs/2504.09702.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. 2025. Scaling test-time compute for LLM agents. *CoRR*, abs/2506.12928.

A Ideation RL Training Formulation

A.1 Step-Level RL Objective

We train the IDEATOR's policy, $\pi_\theta(a|s)$, which is parameterized by a small language model with parameters θ. Following recent work, we adopt a step-level RL paradigm for efficient training. The objective is to find the optimal parameters θ^* that maximize the expected reward from a single-step rollout:

$$\theta^* = \arg\max_\theta \mathbb{E}_{s_t \sim d^{\pi_e}, \alpha \sim \pi_\theta(\cdot|s_t)}[R_t(\alpha)]$$

where d^{π_e} represents the state distribution of the expert coding agent. This stepwise objective is efficient as it allows us to reuse an offline buffer of trajectories and avoids costly multi-step rollouts.

A.2 GRPO Loss Function

We formulate our GRPO training loss function (Shao et al., 2024) as follows, while removing the KL divergence term from the loss following recent practice (Yu et al., 2025).

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{\substack{s_t \sim d^{e,c} \\ \{\alpha_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|s_t)}} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|\alpha_i|} \sum_{j=1}^{|\alpha_i|} \min\left(r_{i,j}(\theta) A_{i,j}, \text{clip}(r_{i,j}(\theta), 1-\epsilon, 1+\epsilon) A_{i,j}\right) \right],$$

$$\text{with } r_{i,j}(\theta) = \frac{\pi_\theta(\alpha_{i,j}|s_t, \alpha_{i,<j})}{\pi_{\theta_{\text{old}}}(\alpha_{i,j}|s_t, \alpha_{i,<j})}$$

$$\text{and } A_{i,j} = \frac{R_t(\alpha_i) - \text{mean}(\{R_t(\alpha_i)\}_{i=1}^G)}{\text{std}(\{R_t(\alpha_i)\}_{i=1}^G)}.$$

B Training Details

B.1 Training Task Selection

For training, we selected 10 tasks spanning diverse modalities—text, image, audio, video, and tabular—whose solutions can typically be executed in under 20 minutes. They are:

- osic-pulmonary-fibrosis-progression
- multi-modal-gesture-recognition
- chaii-hindi-and-tamil-question-answering
- mlsp-2013-birds
- google-quest-challenge
- tgs-salt-identification-challenge
- tweet-sentiment-extraction
- spaceship-titanic
- jigsaw-unintended-bias-in-toxicity-classification
- AI4Code

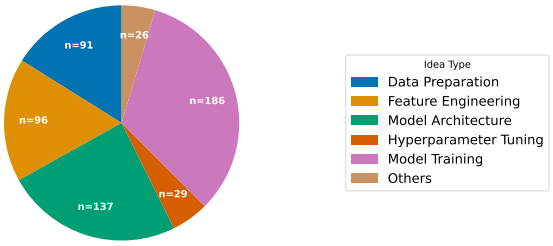

Figure 3: Distribution of idea types, aggregated across Qwen3-8B, Qwen3-8B-RL and Claude Sonnet 3.5 as IDEATORS, paired with Claude Sonnet 3.5 as the implementer.

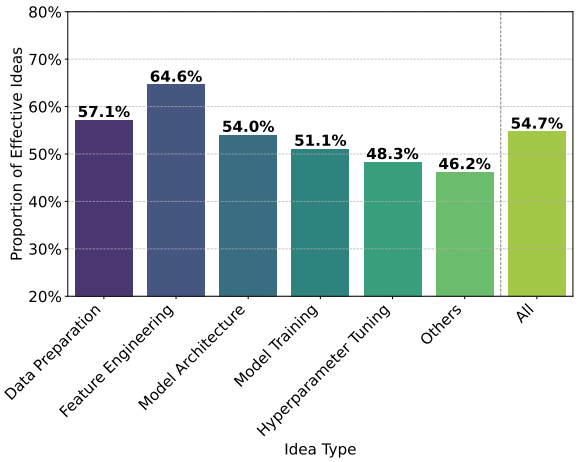

Figure 4: Proportion of effective ideas in each idea type, aggregated across Qwen3-8B, Qwen3-8B-RL and Claude Sonnet 3.5 as IDEATORS, paired with Claude Sonnet 3.5 as the Implementer. We define an effective idea as the performance of the refined ML solution according to the idea is better than before.

B.2 Training Hyperparameters

We use VeRL (Sheng et al., 2025) as our RL training framework. We train the Qwen3-8B model with LoRA (Hu et al., 2022) using a rank of 32 for one epoch on a single node (8x A100 40GB GPUs) with a batch size of 16 and a rollout size (G) of 8 candidate ideas per prompt. We save checkpoints every 10 steps, selecting the best one based on validation reward. The reward calculation is distributed across 16 nodes, each with 8 A10G (24GB) GPUs, where each GPU is dedicated to running the solution for a single candidate idea. The total training time is approximately 52 hours.

C Distribution and Effectiveness of Idea Types

Figure 3 shows the overall distribution of ideas. The most frequent suggestions concerned **Model Training**, followed by **Model Architecture** and **Feature Engineering**. However, frequency alone does not imply effectiveness. We define an idea as *effective* if it improved the ML solution's performance. As shown in Figure 4, **Feature Engineering** had the highest success rate (64.6%), followed by **Data Preparation** (57.1%). In contrast, **Model Training**, despite being the most frequent category, was only the fourth most effective (51.1%), while **Hyperparameter Tuning** was least effective (48.3%). This highlights a central insight: *the most impactful interventions often concern data and features rather than the model itself.*

D Prompt for the IDEATOR

The specific instructions provided to the IDEATOR agent are detailed below. This prompt is designed to guide the model in generating high-quality, actionable ideas, while also specifying the required output format.

Idea Generation Prompt

You are a machine learning expert. Another AI agent is struggling to improve the performance of its machine learning solution. Your task is to analyze the agent's progress and provide the most effective algorithmic idea that can significantly improve the performance.

You will be provided with the agent's history, including its previous attempts and the full trajectory of its actions.

{query}

TRAJECTORY:
{step_trace}

Instruction
Evaluate the current trajectory, pick the **single highest-impact next action**, and then output **exactly three items** in this format:

ANALYSIS_ON_CURRENT_PROGRESS:
<Briefly state whether to keep refining the present approach or revert to a prior solution and pursue a new path.>

ACTION:
<One imperative command or code block the agent must execute next.>

RATIONALE:
<Concise justification for why this action is optimal right now.>

Do **not** list alternatives, background, or extra commentary. Output nothing beyond those three items.

E Prompt for Idea Type Classification

We utilize the following prompt to instruct Claude Sonnet 4 in categorizing the generated ideas. The prompt includes detailed definitions for each idea type and provides the model with rubrics to distinguish between categories.

Idea Type Classification Prompt

You are a machine learning expert. Another AI agent is working on a task and has encountered a problem. Your goal is to analyze a proposed idea for improving a machine learning model and classify it into one of the specific categories defined below.

{trace}

Proposed Idea

{idea}

Your Task:
- Review the trajectory to understand the agent's attempts so far to solve the task.
- Consider the problem description and the proposed idea.
- Determine which category (from a predefined list below) best describes the core focus of the idea.
- Provide a brief rationale for your classification.

* Data Preparation: Ideas for handling missing values, correcting errors, removing outliers, or augmenting existing data.
* Feature Engineering: Ideas for creating new predictive features, transforming existing features (e.g., scaling, encoding), or selecting the most important ones.
* Model Architecture: Ideas for switching to a completely different type of model (e.g., from XGBoost to a Neural Network) or changing the fundamental structure of the current model (e.g., adding/removing layers).
* Hyperparameter Tuning: Ideas for systematically searching for the best model settings (e.g., learning rate, tree depth) to improve performance.
* Model Training: Ideas for changing the training process (e.g., using a different loss function, implementing cross-validation) or using new metrics to evaluate the model.
* Others: Use this category only if the idea clearly does not fit into any of the categories above.

Output Format: output **exactly two items** in this format:

TYPE: <one of "Data Preparation", "Feature

Engineering", "Model Architecture", "Hyperparameter Tuning", "Model Training", or "Others">

RATIONALE: <briefly explain why the idea belongs to the type above>

Output nothing beyond those two items.

F Case Studies on Ideation

F.1 An Effective Feature Engineering Idea

In the *random-acts-of-pizza* task where numerical features showed very low importance (∼1–1.5%), the Ideator suggested creating interaction-based derived features such as `karma_ratio`, `activity_ratio`, `engagement_score`, `text_length`, and `time_of_day`. These features added contextual structure that the raw values lacked, and the Implementer's accuracy improved from 0.60 → 0.64.

Implementer Query

PROBLEM_STATEMENT:
Initial model shows relatively low feature importance for numerical features, need to improve model performance
ATTEMPTS_SO_FAR:
- Created baseline with TF-IDF text features and basic numerical features
- Used RandomForestClassifier with default parameters
- Feature importance shows numerical features each contributing only ∼1-1.5%

GOAL:
Identify ways to improve model performance

Ideator Response

ANALYSIS_ON_CURRENT_PROGRESS:
Keep refining the present approach, as the foundation with text and metadata features is sound but needs optimization.
ACTION:

```
# Create derived features that capture
    interaction effects:
karma_ratio = upvotes_minus_downvotes /
    upvotes_plus_downvotes
activity_ratio = comments_in_raop /
    total_comments
engagement_score = number_of_comments *
    karma_ratio
text_length = len(request_text)
time_of_day = extract hour from
    unix_timestamp
```

RATIONALE:
Raw numerical features show low importance because they lack context - derived features that capture relationships between metrics will better represent user credibility and request quality patterns.

F.2 An Ineffective Hyperparameter Tuning Idea

In the *whale-categorization-playground* task, the Implementer had already reached 72.31% MAP@5 with a ViT model and combined Focal+ArcFace loss. The Ideator proposed further tuning ArcFace parameters (scale, margin, alpha, gamma), but this yielded no benefit and even reduced MAP@5 (0.30 → 0.21). This reflects a common pattern: when the model is already near its capacity and only minor hyperparameters remain, ideation provides limited gains.

Implementer Query

PROBLEM_STATEMENT:
Need to further improve MAP@5 score with current ViT + Combined Loss approach that achieved 72.31% accuracy
ATTEMPTS_SO_FAR:
- Used EfficientNet-B0/B3 with augmentations (∼10% accuracy)
- Implemented Focal Loss and ArcFace separately
- Combined approach with ViT + Custom Loss (∼17% accuracy initially)
- Adjusted learning rates and increased epochs (72.31% accuracy)
- Generated valid submission file

GOAL:
Further optimize model performance by either enhancing the current approach or exploring complementary techniques while maintaining the valid submission format

Ideator Response

ANALYSIS_ON_CURRENT_PROGRESS:
The current ViT model with combined Focal and ArcFace loss has achieved the same accuracy as the initial ViT approach,

suggesting the loss function may not be optimally balanced. Further tuning of the loss parameters or exploring advanced regularization techniques could yield improvements.

ACTION:

```
criterion = CombinedLoss(num_classes=len(
    train_dataset.id_to_idx),
    embedding_size=768, scale=60.0,
    margin=0.3, alpha=0.5, gamma=4)
```

RATIONALE:

Increasing the ArcFace scale factor to 60.0 improves the margin between classes, while adjusting alpha and gamma to 0.5 and 4 respectively enhances focus on hard examples. This directly addresses class imbalance and improves feature discrimination, which is critical for the task.