Extended Abstract

Motivation Reinforcement learning has become a central mechanism for converting latent language-model capability into reliable task behavior. In reinforcement learning from human feedback (RLHF), human preference data trains reward models that support instruction following and alignment (Ouyang et al., 2022). In reinforcement learning from verifiable rewards (RLVR), automatically checkable outcomes replace much of this supervision, making domains such as mathematics, code, and software engineering attractive because rewards can be scaled without dense human labeling (OpenAI, 2024; Shao et al., 2024; Guo et al., 2025; Jimenez et al., 2024). Machine learning engineering (MLE) is also verifiable: a generated solution can be executed and scored on held-out data. However, MLE verification is more expensive than checking a short answer or running a small unit-test suite. A rollout must produce code, execute in an environment, train or apply a model, create a schema-valid submission, and then be graded. This creates both a cold-start problem, because base models often fail before producing valid submissions, and a verifier-cost problem, because successful rollouts can require nontrivial execution time.

Method We reproduce and extend the setup of Yang et al. (2026), who train MLE agents with reinforcement learning using held-out performance as a verifiable reward, partial environment instrumentation for denser feedback, and duration-aware policy-gradient updates for variable-cost actions. We focus on `random-acts-of-pizza`, a Kaggle text-classification task, following the MLE-Bench and MLE-Dojo framing of executable ML-engineering environments (Chan et al., 2025; Qiang et al., 2025). A policy generates Python solution code, the environment executes the code, and the reward combines final verifier score with partial progress markers. For invalid submissions we use a partial-instrumentation reward, $r = -10 + 0.1m$, where m counts workflow markers such as importing packages, loading data, defining a model, training, predicting, and writing a submission. For valid submissions with finite verifier scores, r is the verifier score. We use Group Relative Policy Optimization (GRPO), a critic-free alternative to Proximal Policy Optimization (PPO) that computes relative advantages within sampled rollout groups (Schulman et al., 2017; Shao et al., 2024). We also study replay variants and reward models as mechanisms for stabilizing learning and amortizing verifier cost.

Implementation The policy experiments use Qwen2.5-Coder as the code-generation base model (Hui et al., 2024), with supervised fine-tuning (SFT) followed by GRPO. Each training update samples groups of rollouts for the same task prompt and executes generated solutions in the MLE environment. We evaluate cold-start validity, Qwen SFT + GRPO training dynamics, verifier execution cost, replay variants, offline reward-model pair ranking, and on-policy reward-model calibration. The reward model is trained with pairwise preferences over Kaggle scored solutions, using a Bradley-Terry-style pairwise objective (Bradley and Terry, 1952), and then refreshed with low-rank adaptation (LoRA) on verified on-policy rollout pairs (Hu et al., 2021).

Results Cold-start validity is sparse but nonzero: the base Qwen probe produced 2 valid submissions out of 128 rollouts. After SFT and GRPO, the final Qwen run reaches 128/128 valid training submissions and 16/16 valid validation submissions at step 100, with a final validation score of 0.62898 on `random-acts-of-pizza`. Replay variants affect operational stability, but the tested variants do not clearly break the score plateau around the learned TF-IDF/logistic-regression solution template. Runtime diagnostics show why verifier cost matters: as valid-submission rate rises, more rollouts run far enough to train models and produce predictions, increasing the average execution time required for each RL update.

Discussion and Conclusion The reward-model results separate two regimes. Offline Kaggle reward modeling learns real ranking signal, with about 0.628 held-out pair accuracy, but transfers poorly to on-policy Qwen rollouts, where held-out pair accuracy is 0.521 and Spearman correlation is negative. A small on-policy LoRA refresh improves held-out pair accuracy to 0.693 and Spearman correlation to 0.562. These results suggest that direct RLVR can reliably train validity and submission formation on a narrow MLE task, while improving true MLE score among already-valid solutions is harder. We use reward-model-guided GRPO (RM-GRPO) to refer to alternating verifier-scored updates with updates scored by a calibrated reward model. Our RM-GRPO evidence remains preliminary; the completed positive result is on-policy reward-model calibration.

Reward-Model-Calibrated Reinforcement Learning from Verifiable Rewards for Machine Learning Engineering

Siddharth Sachdeva
Department of Computer Science
Stanford University
TODO: email

Abstract

Machine learning engineering (MLE) is a natural target for reinforcement learning from verifiable rewards (RLVR): generated code can be executed and scored on held-out data. Yang et al. recently showed that reinforcement learning for MLE agents can outperform static prompting by combining verifiable held-out rewards, partial environment instrumentation, and duration-aware updates. We reproduce and extend this line of work on `random-acts-of-pizza`, a Kaggle text-classification task, using Qwen2.5-Coder, partial progress rewards for invalid code, and Group Relative Policy Optimization (GRPO) rather than PPO-style critic-based optimization. After supervised fine-tuning, Qwen + GRPO learns to produce valid submissions reliably, reaching 128/128 valid training rollouts and 16/16 valid validation rollouts with final validation score 0.62898. Replay variants affect stability but do not clearly escape the learned score plateau. We then study reward models as a path to verifier amortization. An offline Kaggle pairwise reward model achieves 0.628 held-out pair accuracy but transfers poorly to on-policy rollouts; on-policy low-rank adaptation (LoRA) calibration improves held-out pair accuracy from 0.521 to 0.693 and Spearman correlation from -0.058 to 0.562. These results show that RLVR can solve the validity regime for a narrow MLE task, while calibrated reward models are a promising but not yet fully demonstrated route to cheaper score optimization.

1 Introduction

Reinforcement learning is increasingly used as a post-training mechanism for making large language models (LLMs) reliable in domains where raw next-token prediction is not enough. In reinforcement learning from human feedback (RLHF), reward models trained from human preferences were central to making instruction-tuned models useful for broad deployment (Ouyang et al., 2022). More recently, reinforcement learning from verifiable rewards (RLVR) has shown that when outcomes are automatically checkable, learning can scale with less human supervision. Reasoning models such as OpenAI o1 and open work such as DeepSeekMath and DeepSeek-R1 demonstrate that verifiable rewards can drive gains in math, code, and reasoning-heavy tasks (OpenAI, 2024; Shao et al., 2024; Guo et al., 2025).

Machine learning engineering is a compelling but difficult RLVR domain. Any MLE task with a specified evaluation procedure has a natural verifier: a generated solution can be executed and scored on held-out data. In the Kaggle-style setting we study, this means the solution must produce a schema-valid submission and receive a score from the task grader. This makes MLE unlike open-ended text generation, where reward often depends on preference labels. At the same time, MLE differs from many coding tasks. A correct-looking solution must parse, run, load data, construct features,

train or apply a model, produce the exact submission schema, and complete within execution limits. Verification cost is therefore coupled to the generated ML pipeline.

MLE-Bench made this setting concrete by evaluating agents on Kaggle-sourced machine learning tasks (Chan et al., 2025). MLE-Dojo extends the setting with an interactive and executable environment for training and evaluating MLE agents (Qiang et al., 2025). Most MLE-Bench progress has focused on prompting, scaffolds, search, and agent loops. The most directly relevant exception is Yang et al. (2026), who show that smaller language-model agents can improve through reinforcement learning on MLE tasks and identify two challenges that are central to our work: variable-duration code execution and sparse feedback from held-out performance alone.

This paper studies a narrower reproduction and extension of that result: can we use critic-free RLVR to train an LLM policy to produce valid MLE submissions, and can reward models help reduce verifier cost once rollouts become valid-looking? We focus on `random-acts-of-pizza`, a Kaggle-style text classification task. Our contributions are:

- We reproduce the core MLE RLVR effect of Yang et al. (2026) with supervised fine-tuned Qwen2.5-Coder and Group Relative Policy Optimization (GRPO), using partial instrumentation rewards and official verifier scores.
- We quantify the cold-start and execution-cost issues that make MLE RL different from short-answer RLVR.
- We compare replay variants and show that replay matters operationally, but our tested variants do not clearly escape the score plateau.
- We evaluate offline and on-policy reward models, showing that offline Kaggle ranking signal is real but on-policy calibration is necessary before using a reward model for rollout ranking.

2 Related Work

RLHF and reward modeling RLHF trains language models to optimize learned human-preference rewards rather than only supervised targets or likelihood (Ouyang et al., 2022). This introduced a practical recipe of supervised fine-tuning, reward modeling, and policy optimization, usually with PPO (Schulman et al., 2017). Our reward-model experiments inherit the same broad idea, but the supervision source differs: pair labels come from measured MLE scores or ranks rather than human preference judgments.

RLVR for reasoning and code RLVR replaces human preference labels with automatically verifiable outcomes. OpenAI o1 is a closed model demonstration of reasoning improvement through reinforcement learning (OpenAI, 2024). DeepSeekMath introduced GRPO as a memory-efficient PPO variant that removes the learned value model and estimates relative advantage within sampled groups (Shao et al., 2024). DeepSeek-R1 further popularized RL for reasoning and open recipes for cold-start and multi-stage RL (Guo et al., 2025). Software engineering benchmarks such as SWE-bench also rely on executable verification, typically through issue-specific tests (Jimenez et al., 2024). MLE verification is related but often more expensive because generated code may train models or process large datasets.

Machine learning engineering benchmarks MLE-Bench evaluates agents on 75 Kaggle-sourced tasks and compares automated agents against human-level competition outcomes (Chan et al., 2025). MLE-Dojo provides an executable Gym-style environment with over 200 Kaggle challenges and supports both supervised fine-tuning and reinforcement learning (Qiang et al., 2025). We use this line of work as the environment framing rather than proposing a new benchmark. Our scope is intentionally narrower: a focused study of direct GRPO on one task and reward-model calibration for verifier amortization.

RL for MLE agents Yang et al. (2026) provide the closest prior work. They formulate MLE agents as reinforcement-learning policies over generated plans and code, use held-out performance as a verifiable reward, and introduce two key engineering ideas: duration-aware gradient updates for variable-time actions and environment instrumentation for partial credit when generated programs fail before final scoring. Our work follows their environment and reward motivation but changes

the optimizer to critic-free GRPO, focuses on a narrower reproduction on `random-acts-of-pizza`, and adds reward-model calibration experiments for verifier-cost amortization.

3 Method

Environment and rollout Each episode presents the policy with a prompt describing the `random-acts-of-pizza` task and the requirement to produce a valid `submission.csv`. The policy is initialized from Qwen2.5-Coder, a code-specialized open model family (Hui et al., 2024). The model generates Python code, which is parsed and executed in the task environment. The environment records whether code is present, whether execution succeeds, whether a submission exists, whether the submission schema is valid, whether a finite verifier score is available, and how long execution takes. The main GRPO run uses 128 training rollouts per update and 16 validation rollouts at evaluation points.

Instrumented reward We use a partial-instrumentation reward to avoid giving every invalid program the same signal. If a rollout produces a valid submission with a finite verifier score, the reward is the verifier score:

$$r = s_{\text{verifier}}.$$

Otherwise, the reward is

$$r = -10 + 0.1m,$$

where m is the number of workflow progress markers observed. The markers represent coarse MLE progress: importing packages, loading data, defining a model, training or fitting, predicting, and creating a submission. This reward is not intended to measure final solution quality among invalid rollouts. Its purpose is to provide a gradient before the policy can reliably reach verifier-scored submissions.

GRPO We use GRPO, following the critic-free idea introduced in DeepSeekMath (Shao et al., 2024). For each prompt, the trainer samples a group of rollouts and computes relative advantages within the group. The update increases the likelihood of outputs whose rewards are high relative to same-prompt alternatives. The practical motivation is memory: avoiding a learned critic reduces overhead in a setting where rollout execution is already expensive.

Replay variants Because valid submissions are initially sparse, we study replay mechanisms that reintroduce useful prior rollouts into later GRPO updates. In replay, some examples in the current update are not newly sampled rollouts; they are previously generated submissions drawn from a buffer. The intent is to prevent early valid submissions from disappearing from the training signal before the policy can reliably regenerate them. Following the replay motivation in Yang et al. (2026), we replay from verifier-valid submissions. We tested round-robin replay, top-k replay, stable-uniform replay, and previous-step replay, with $k = 8$ and minimum reward threshold 0.5 in the main Qwen manifests. A key implementation detail is the threshold comparison: replaying every valid submission admits exact-0.5 baseline submissions, while strict reward > 0.5 filters them out.

Reward models The offline reward model is trained on Kaggle scored solution pairs from the same competition: the chosen solution has better rank, score, or percentile than the rejected solution. Training uses a pairwise Bradley-Terry-style objective over reward differences (Bradley and Terry, 1952). We then evaluate the model on held-out Kaggle pairs and on verified on-policy Qwen rollouts. For on-policy calibration, we construct pairs from true verifier scores, split into train and held-out sets, and fine-tune the offline model with low-rank adaptation (LoRA) (Hu et al., 2021).

4 Results

4.1 Cold Start Is Sparse But Nonzero

Table 1 shows why we selected Qwen2.5-Coder for the RL experiments. Among the models we tested under roughly 4B parameters, Qwen2.5-Coder-3B-Instruct was the only model that produced valid submissions on both evaluated tasks, with 8/1024 valid Pizza submissions and 3/128 valid Plant Pathology submissions. Other small models either produced no valid submissions or produced

Table 1: Cold-start valid-submission rates for models evaluated on both Pizza and Plant Pathology. Bold entries indicate nonzero valid submissions.

Model	Pizza	Plant Pathology
Qwen3-4B-Instruct	0/1024 (0.0%)	0/128 (0.0%)
Qwen2.5-Coder-3B-Instruct	**8/1024 (0.8%)**	**3/128 (2.3%)**
Nemotron-Mini-4B-Instruct	**21/1024 (2.1%)**	0/128 (0.0%)
Phi-4-mini-instruct	**42/1024 (4.1%)**	0/128 (0.0%)

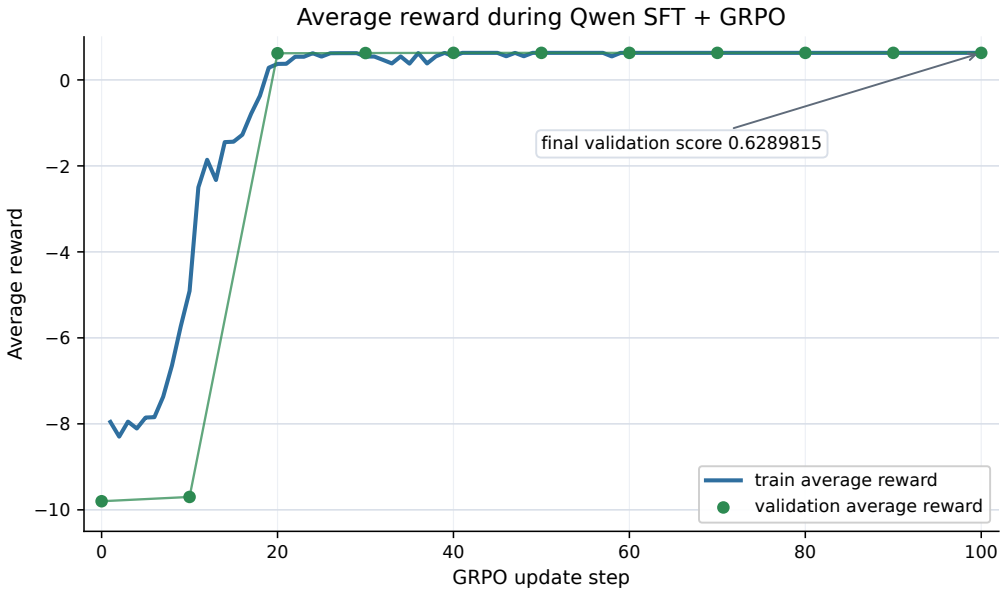

Figure 1: Qwen SFT + GRPO training curve on `random-acts-of-pizza`.

valid submissions on only one task. This cross-task nonzero validity was still sparse, but it gave enough signal for RL to be meaningful: the policy could occasionally reach an executable, schema-valid solution, while partial instrumentation could provide additional gradient before verifier-scored submissions became reliable. In the smaller base Qwen probe used for the main task, Qwen produced 2 valid submissions out of 128 rollouts, for a 1.6% valid-submission rate, and a Qwen agent-prompting probe produced 1 valid submission out of 96 rollouts.

4.2 Qwen SFT + GRPO Learns Reliable Validity

The completed Qwen SFT + GRPO run reaches full validity by the end of training. At step 100, the final training batch has 128/128 valid submissions and the final validation set has 16/16 valid submissions. The final validation score is 0.6289815. Across the whole run, 11,595 of 12,800 training rollouts are valid, a valid rate of 90.6%.

Qualitatively, the trained policy converges toward a reliable text-classification template using TF-IDF features and logistic regression. Example traces show the contrast with base behavior: base Qwen often writes plausible-looking code that fails at execution or submission creation, while trained rollouts repeatedly execute cleanly and receive the same nontrivial verifier score. This is a real RLVR success, but it is also a limited one: the policy learns to reliably express a reasonable solution template rather than discovering a sequence of substantially stronger modeling strategies.

4.3 Verifier Cost Remains a Bottleneck

Verifier cost is a central difference between MLE RLVR and RLVR for many math or coding tasks. In math, verification can often be a string or symbolic check; in coding, it is often a bounded unit-test run.

Table 2: Accuracy comparison on `random-acts-of-pizza`.

Method	Accuracy
Qwen-AIDE	0.57
Qwen RLVR	0.63
Original Paper	0.66

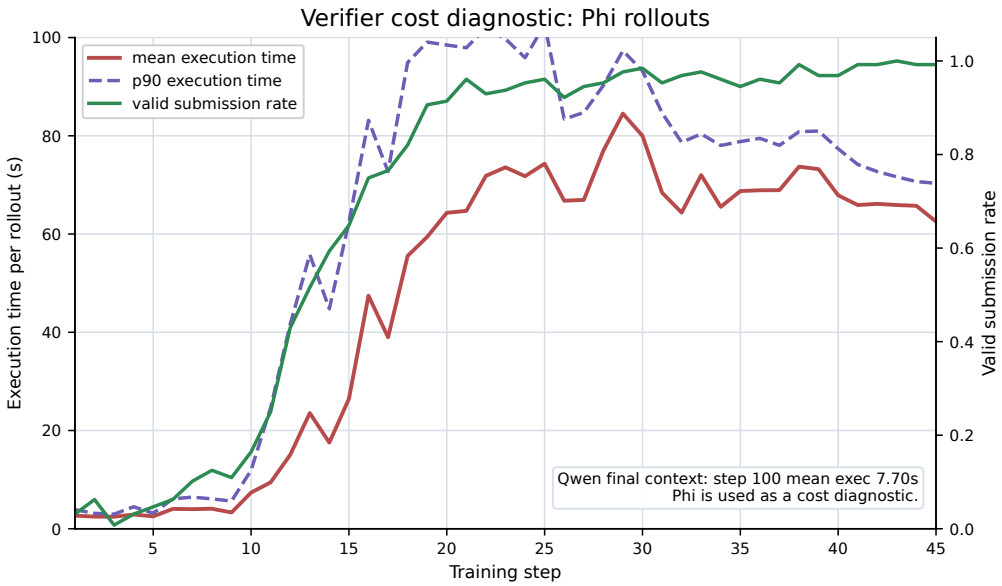

Figure 2: Verifier/execution-cost diagnostic. As the policy produces more valid submissions, more rollouts run training and prediction pipelines, so average execution time per update rises.

In MLE, a valid-looking rollout may need to load data, build features, train a model, run validation or prediction, and write a submission before the verifier can assign a final score.

This cost becomes larger as the policy improves. Early invalid rollouts often fail before full training and validation. As valid-submission rate rises, more rollouts run far enough to execute the expensive parts of the ML pipeline, so the average time per training step increases. Figure 2 illustrates this pattern: valid-submission rate rises from 3.1% to 91.4%, while mean rollout execution time rises from 2.66s to 64.33s. This is the main motivation for reward-model amortization: once rollouts are mostly valid-looking, scoring every candidate with the full verifier can dominate the cost of RL.

4.4 Replay Affects Stability But Not the Plateau

Replay matters operationally because the training distribution can otherwise lose rare useful valid submissions before the policy has learned to produce them consistently. The replay recipe in Yang et al. (2026) replays from valid submissions, and our initial implementation followed that idea by admitting all valid submissions into replay. On `random-acts-of-pizza`, however, this included exact-0.5 submissions, which are valid but correspond to weak baseline behavior. The old previous-step replay run therefore converged to a lower validation score of 0.567 rather than the stronger TF-IDF/logistic-regression template.

The main replay finding is that validity alone is not a good replay filter for this task. Filtering replay candidates with strict reward > 0.5 removes the exact-baseline valid submissions and changes the outcome: stable-uniform replay reaches 0.628982, and previous-step replay reaches 0.630390. Table 3 summarizes this threshold effect. The replay variants still do not clearly solve the final score plateau, but the threshold audit gives a concrete implementation lesson: in MLE RL, "valid" submissions can be too weak to replay blindly.

Table 3: Replay ablation showing the threshold effect: replaying all valid submissions admits exact-0.5 baselines, while strict reward > 0.5 filters them out.

Replay filter	Selection	Steps	Final val. score	Note
≥ 0.5	stable uniform	–	–	old-filter uniform run not present in checked-in summary
≥ 0.5	previous step	1	0.567049	old lower-scoring previous-step run
> 0.5	stable uniform	85	0.628982	strict filter / current implementation
> 0.5	previous step	100	0.630390	strict filter / best previous-step run

The manifests record `min_reward=0.5` but not the comparison operator; comparator labels use the replay-code audit described in the figure source notes. The old-filter stable-uniform metric was not present in the checked-in summary, so it is not filled in here.

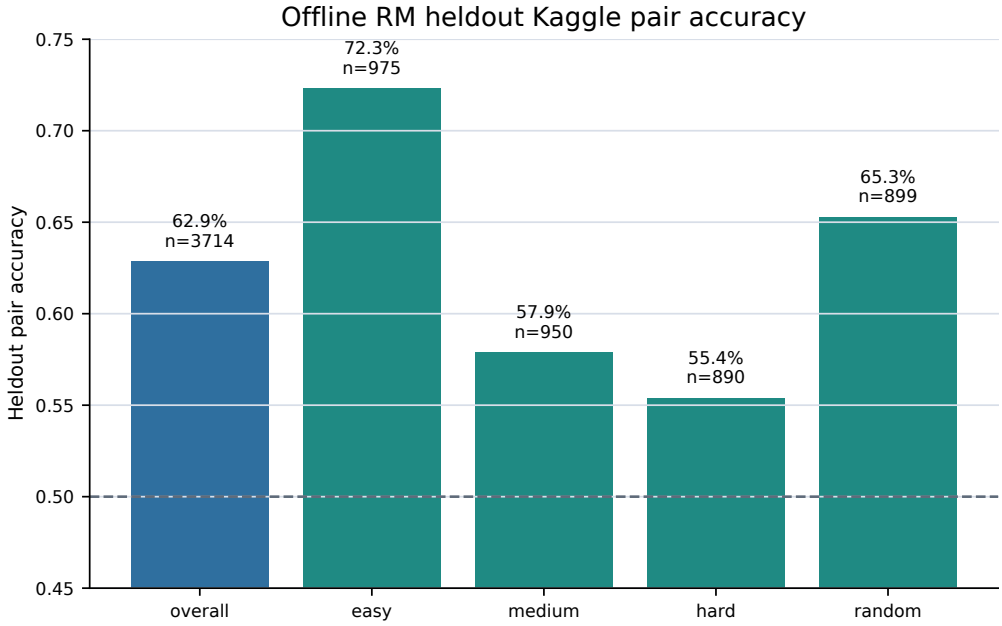

Figure 3: Offline reward-model evaluation on held-out Kaggle pair-ranking data.

4.5 Offline Reward Models Learn Kaggle Ranking Signal

The offline reward model has nontrivial held-out Kaggle ranking signal. The main held-out evaluation reports overall pair accuracy of 0.6287 over 3,714 pairs. Accuracy is highest on easy pairs, at 0.7231, and lower on medium and hard pairs, at 0.5789 and 0.5539. A second held-out evaluation reports a similar overall accuracy of 0.6278 over 2,437 pairs, with bucket accuracies 0.6916 easy, 0.6063 medium, 0.5751 hard, and 0.6287 random. These numbers are strong enough to justify reward-model experiments, but they do not establish that the model will rank on-policy rollouts correctly.

4.6 On-Policy Calibration Is Necessary

Direct transfer from offline Kaggle pairs to on-policy Qwen rollouts is weak. On held-out verified-score pairs from the on-policy distribution, the base offline reward model has pair accuracy 0.5215 over 512 pairs, Spearman correlation -0.0581, Kendall tau-b -0.0367, and top-10% overlap 0.0.

After a small on-policy LoRA refresh, held-out pair accuracy improves to 0.6934. Spearman correlation improves to 0.5619, Kendall tau-b to 0.3852, and top-10% overlap to 0.5714. The refresh uses 512 on-policy train pairs and 512 held-out on-policy pairs constructed from verified Qwen rollouts, with LoRA rank 8, LoRA alpha 16, learning rate 10^{-5}, and 200 training steps. This is the clearest positive reward-model result: offline reward-model signal is useful, but it must be calibrated on verified on-policy rollouts before it can be trusted for model-generated solutions.

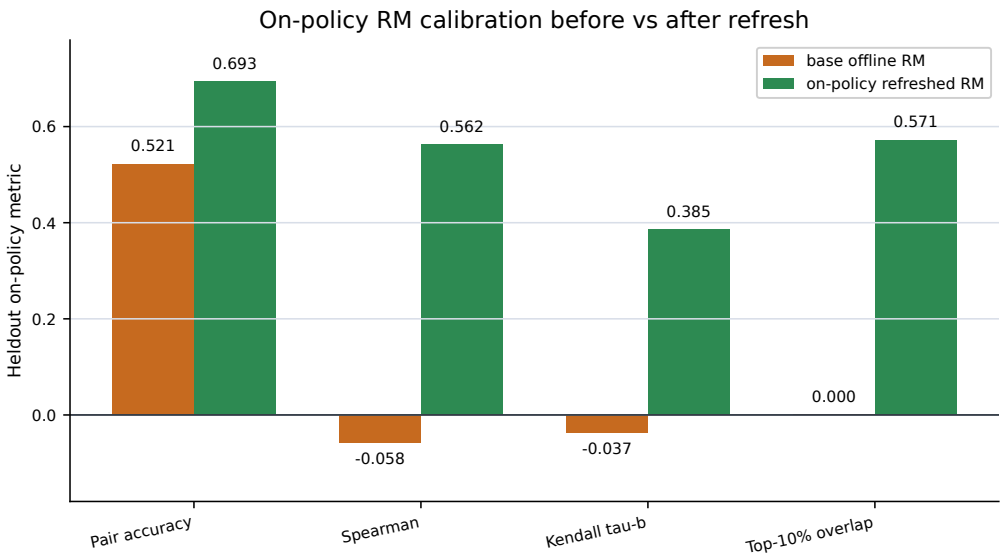

Figure 4: On-policy reward-model calibration before and after LoRA refresh.

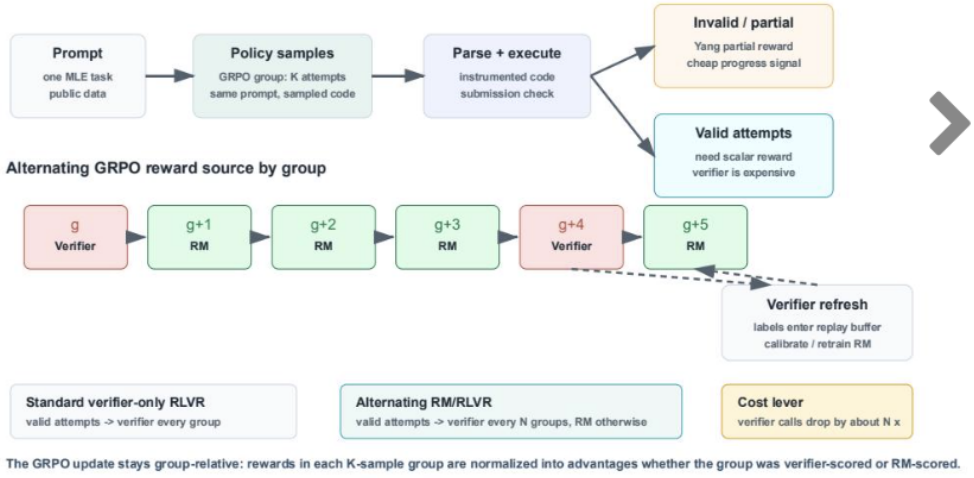

Figure 5: Planned reward-model-guided GRPO loop: after calibrating a reward model on verified on-policy rollouts, the reward model can score some valid-looking rollouts in place of the expensive verifier.

4.7 RM-GRPO Status

RM-GRPO refers to the alternating training design in Figure 5: after obtaining a reward model with good on-policy calibration, some phases use official verifier rewards and other phases use the frozen calibrated reward model to score valid-looking rollouts. We were trying to run this loop after the reward-model calibration result in Section 4.6, but we did not obtain a complete successful RM-GRPO training run. The available reward-quality report shows that the alternating pipeline passed basic acceptance checks, with valid-submission rate 0.810 and verifier-refresh reward source count 2,798 out of 3,456 traces, but full training repeatedly ran into out-of-memory failures that we could not fix before running out of time. The completed positive result is therefore reward-model calibration, not reward-model-guided policy improvement. We hope to finish the RM-GRPO experiments and turn this project into a research paper.

5 Discussion

Validity is easier than MLE score improvement The clearest success is reliable validity. Base models rarely produce valid submissions, but Qwen SFT + GRPO learns to do so almost perfectly on the target task. This is useful because validity is a prerequisite for any meaningful MLE score optimization. However, validity is not the same as high MLE performance. The policy largely converges to a repeated valid template and plateaus around score 0.629.

Partial instrumentation is essential The reward formula is coarse, but it addresses the cold-start problem. Without partial progress rewards, most early rollouts would receive indistinguishable invalid outcomes. With markers, the policy receives some signal for learning the structure of an MLE solution even before it reliably creates valid submissions.

Replay is a training-system detail, not a solved algorithmic story Replay can preserve useful valid outputs, but it can also preserve weak baselines. The exact threshold comparison matters because reward 0.5 corresponds to a flat baseline submission in this task. Strictly excluding exact-0.5 replay rows improved the quality of the replay buffer, but none of the tested replay variants clearly solved the score plateau.

Reward models need on-policy calibration The offline reward-model result is encouraging but insufficient. Kaggle solution pairs teach the model a general notion of solution quality, yet on-policy rollouts are distributed differently: many are near-duplicates, template variants, or valid-looking solutions with small score margins. The calibration results suggest a practical strategy for verifier amortization: periodically verify a subset of on-policy rollouts, refresh the reward model on these pairs, and use the calibrated model only within the distribution it has recently seen.

Limitations This study is narrow. The main RL result is on one task, `random-acts-of-pizza`, and the learned policy appears to exploit a stable text-classification template. We do not yet demonstrate broad generalization across MLE-Bench tasks. The RM-GRPO result is preliminary. Finally, the verifier-cost result is a motivation for amortized scoring rather than a complete cost model for all MLE tasks; runtime will depend on the generated pipeline, dataset size, and task-specific grader.

6 Conclusion

We find that GRPO with partial instrumentation can reproduce the core RLVR effect for a narrow MLE task: a Qwen2.5-Coder policy learns to produce valid `random-acts-of-pizza` submissions reliably and reaches a nontrivial score. This validates the use of verifier-based RL for the validity regime of MLE. The harder regime is improving true MLE score among already-valid submissions. Offline reward models provide useful Kaggle ranking signal but transfer poorly to on-policy rollouts; a small on-policy LoRA refresh substantially improves ranking quality. The next step is to close the loop by demonstrating that calibrated RM-GRPO improves score or verifier efficiency under final official verification.

7 AI Tools Disclosure

We used OpenAI Codex as an assistant during this project. Codex helped with infrastructure work, data pipelines and processing, understanding the VERL repository, getting feedback on the initial report outline, and revising the writing. The authors were responsible for the experimental design, interpretation of results, final claims, and final submitted text.

Changes from Proposal In the proposal, we initially planned to focus on comparing supervised fine-tuning against a non-SFT baseline. As the project developed, the central bottleneck became verifier cost: once the policy started producing valid-looking MLE submissions, each RL step required running more expensive training and validation pipelines. We therefore evolved the project toward reward models for verifier amortization, studying whether an offline Kaggle reward model could be calibrated on verified on-policy rollouts and eventually used to replace some full verifier calls during RM-GRPO.

References

Ralph Allan Bradley and Milton E. Terry. 1952. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika* 39, 3/4 (1952), 324–345. doi:10.1093/biomet/39.3-4.324

Jun Shern Chan, Keith Chow, Oliver Jaffe, James Aung, Dan Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Heidy Wijk, Lilian Witten, and Long Ouyang. 2025. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. In *International Conference on Learning Representations*. https://arxiv.org/abs/2410.07095

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025). https://arxiv.org/abs/2501.12948

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685* (2021). https://arxiv.org/abs/2106.09685

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-Coder Technical Report. *arXiv preprint arXiv:2409.12186* (2024). https://arxiv.org/abs/2409.12186

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. In *International Conference on Learning Representations*. https://arxiv.org/abs/2310.06770

OpenAI. 2024. Learning to Reason with LLMs. https://openai.com/index/learning-to-reason-with-llms/.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training Language Models to Follow Instructions with Human Feedback. *Advances in Neural Information Processing Systems* (2022). https://arxiv.org/abs/2203.02155

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. *arXiv preprint arXiv:2505.07782* (2025). https://arxiv.org/abs/2505.07782

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017). https://arxiv.org/abs/1707.06347

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300* (2024). https://arxiv.org/abs/2402.03300

Sherry Yang, Joy He-Yueya, and Percy Liang. 2026. Reinforcement Learning for Machine Learning Engineering Agents. In *International Conference on Learning Representations*. https://openreview.net/pdf?id=mfIbSouoaZ

A Figure Numbering

The report uses the following visual sequence: cold start/model variation table, Qwen GRPO training curve, accuracy comparison table, verifier cost figure, replay ablation table, offline RM evaluation figure, on-policy RM calibration figure, and reward-model-guided GRPO schematic.

B Example Trace Summary

The qualitative traces support the interpretation that the policy learns reliable valid-template production. Base traces include plausible but invalid code that fails at runtime or parsing, plus a rare constant valid baseline with score 0.5. Trained traces at step 100 execute cleanly, write valid submissions, and receive score 0.6289815 with TF-IDF/logistic-regression-style solutions.