MURPHY: Reflective Multi-Turn Reinforcement Learning for Self-Correcting Code Generation in Large Language Models

Chanakya Ekbote*
cekbote@amazon.com
AWS AI LABS

Vijay Lingam†
vjlingam@amazon.com
AWS AI LABS

Sujay Sanghavi
AMAZON

Behrooz Omidvar-Tehrani
AWS AI LABS

Jun (Luke) Huan
AWS AI LABS

Anoop Deoras
AWS AI LABS

Stefano Soatto
AWS AI LABS

Abstract

Reinforcement Learning with Verifiable Rewards (RLVR) has emerged as a powerful framework for enhancing the reasoning capabilities of large language models (LLMs). However, existing approaches such as Group Relative Policy Optimization (GRPO) and its variants, while effective on reasoning benchmarks, struggle with agentic tasks that require iterative decision-making and refinement. We introduce MURPHY, a multi-turn reflective optimization framework that extends GRPO by incorporating iterative self-correction during training. By leveraging both quantitative and qualitative execution feedback, MURPHY enables models to progressively refine their reasoning across multiple steps. Evaluations on code generation benchmarks with model families such as Qwen and OLMo show that MURPHY consistently improves performance, achieving up to a 5% relative gain in pass@1 over GRPO, on similar compute budgets.

1 Introduction

> *"The road to wisdom? Well, it's plain and simple to express:*
> *err and err and err again, but less and less and less."*
> *—Piet Hein*

Reinforcement Learning with Verifiable Rewards (RLVR) has enabled a new generation of language models [13, 4, 16, 20] that demonstrate strong capabilities in complex reasoning tasks, including mathematics, coding, and general problem solving. An emerging body of work investigates large language models (LLMs) as agents for software engineering tasks that require program execution and feedback from the environment [15, 10, 11]. Such agents are typically deployed within an agent scaffold that structures an iterative inference process, enabling them to integrate the reasoning abilities of LLMs with the use of external tools. Their effectiveness hinges on the LLM's ability to incorporate intermediate, inference-time feedback. In software engineering contexts such as code generation [7], this feedback arises naturally from program execution—for instance, through executor logs or unit test outcomes.

*Work performed while the author was a research intern at AWS AI Labs.
†Corresponding author

First Workshop on Foundations of Reasoning in Language Models (NeurIPS 2025).

More recently, RLVR algorithms such as Group Relative Policy Optimization (GRPO) [25] and its variants [24, 22, 18] have become popular approaches for enhancing the reasoning capabilities of large language models (LLMs). GRPO is inherently a single-stage training algorithm: it optimizes model outputs based on a one-shot evaluation signal and does not incorporate iterative, inference-time feedback. While GRPO-trained LLMs achieve measurable improvements on standard reasoning benchmarks in mathematics, coding, and general problem solving, our experiments show that these gains remain modest in agentic settings where structured feedback can be leveraged. To explore this further, we evaluate both base and GRPO-trained variants of Qwen3-1.7B/4B [20] and OLMo-2-7B-Instruct [12] within the Reflexion [15] framework. Across multiple tasks, we find that GRPO-induced improvements are limited (see Tab. 1), suggesting that single-stage optimization alone may not fully prepare models to exploit feedback-driven refinement. Motivated by these findings, we pose the following research question:

How can GRPO be extended to incorporate iterative feedback, and does this lead to greater improvements in agentic frameworks?

To address this, we propose MURPHY, a novel RLVR algorithm that extends GRPO to a multi-turn setting and grounds reasoning in intermediate execution feedback. Extending GRPO to a multi-turn setting is non-trivial, as credit assignment across turns is inherently ambiguous. Our approach begins by generating G rollouts for each prompt in the batch and computing the corresponding rewards and advantage scores. For each rollout that does not achieve the maximum reward, we append the execution feedback (e.g., console logs or unit test results for code generation) and generate an additional G rollouts. This process is repeated for a fixed number of turns. Rewards from the final turn are back propagated to earlier stages if a predefined criterion is satisfied, and the GRPO update is applied to each group within each turn. This baseline formulation can get computationally expensive, as the the total number of turns and total generations in each turn grows. To bound this cost, we explore several pruning strategies to reduce the number of gradient updates at each turn. Our main contributions can be summarized as follows:

Main Contributions.

1. We introduce MURPHY, a novel RLVR algorithm that extends GRPO to multiple stages and incorporates execution feedback for grounded reasoning and improved self-correction. (Subsec. 4.1)

2. We explore several design strategies for pruning rollouts across stages and reducing gradient update costs, making MURPHY more computationally efficient and practically feasible. (Subsec. 4.2)

3. We conduct a comprehensive suite of experiments spanning multiple model families (OLMo, Qwen) and sizes (1.7B, 4B, 7B) across three code generation datasets. Models trained with MURPHY consistently outperform GRPO-trained baselines, achieving up to a 5% improvement in pass@1 (Sec. 5).

2 Related Work

LLM Agents for Software Development. A growing body of work [7, 26] explores the use of LLM agents for programming tasks such as code generation from natural language, bug fixing, and code migration. A key driver of progress in these domains has been inference-time iterative frameworks [15, 10], which exploit execution feedback to generate self-reflections and apply search-based strategies (e.g., BFS, MCTS) for refining candidate solutions. Beyond code generation, researchers have extended this paradigm to broader software engineering workflows [21, 17], where agents are scaffolded to invoke external tools, execute commands, process environment feedback, and plan actions accordingly. While these methods highlight the value of iterative feedback and scaffolding, they primarily operate at the inference level. Our work is complementary: rather than improving the scaffold, we focus on training strategies that enhance the reasoning and self-correction capabilities of LLMs themselves, thereby strengthening the foundations on which agentic coding frameworks are built.

Reinforcement Learning with Verifiable Rewards for LLM Reasoning. Post-training and fine-tuning LLMs with reinforcement learning has become a popular strategy for enhancing reasoning capabilities and aligning outputs with desired targets. The introduction of GRPO [25] renewed interest in RL as an efficient alternative to PPO [14], offering competitive performance with significantly lower computational cost. Subsequent variants of GRPO [24, 22, 23] aim to improve training stability and convergence. However, these methods remain restricted to the single-turn setting, where models are optimized to complete tasks in one step, often at the expense of iterative refinement. Beyond single-turn training, multi-turn RL approaches have also been explored, including value-based, policy-based [8], and model-based methods. [6] proposed μCODE to solve multi turn code generation with single step reward. However, this work requires learning a verifier to score the generated code. The work most closely related to ours is RLEF [5], which grounds code LLMs in execution feedback and iteratively refines generations using PPO. While effective, RLEF requires a separate value function implemented as an additional LLM, leading to significantly higher computational cost and complexity. In contrast, our method, MURPHY, achieves the same goal of grounding models in execution feedback and enabling iterative refinement, but does so by extending GRPO to multi-turn training while preserving its simplicity and efficiency.

3 Background: GRPO

Group Relative Policy Optimization [25] (GRPO) is an adaptation of the Proximal Policy Optimization (PPO) framework aimed at achieving more efficient and stable policy updates in LLM fine-tuning. Unlike PPO, which relies on a learned value function (critic), GRPO generates a set of G candidate responses for each input, forming a *response group*. The method then evaluates and scores each candidate, estimating advantages by normalizing rewards within the group, subtracting the group's mean reward and dividing by its standard deviation, to produce relative, standardized advantage values. Moreover, as in PPO, GRPO can include an additional penalty term to constrain the updated policy π from deviating excessively from the old policy π_{old}. This is typically implemented via the Kullback-Leibler (KL) divergence, for maintaining stability during updates.

We denote the model policy by $\pi(\cdot \mid \cdot)$. Let G be the number of generations, $\mathcal{P}(Q)$ the distribution over input prompts/questions Q, and O the output space. For the i-th generation in a group, $o_i \in O$ denotes the entire generation trajectory, $o_{i,t}$ the t-th token, and $o_{i,<t}$ all tokens up to (but not including) token t. We let $\hat{A}_{i,t}$ denote the advantage of token t for the i-th generation within a group. Note that $D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})$ denotes the KL divergence between the current policy and the reference policy, computed over all tokens in the generated sequences. The GRPO objective can be written as follows:

Definition 1. (GRPO Objective)

$$\mathcal{J}(\theta) = \mathbb{E}_{q\sim\mathcal{P}(Q),\ \{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(O|q)} \Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\min\Big(\frac{\pi_\theta(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})}\hat{A}_{i,t},$$
$$\text{clip}\left(\frac{\pi_\theta(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})},\, 1-\epsilon,\, 1+\epsilon\right)\hat{A}_{i,t}\Big)\Bigg] - \beta D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})$$

4 Proposed Approach

In this section, we present our proposed approach, MURPHY. While the framework is applicable to a range of RLVR algorithms such as PPO and RLOO, we focus primarily on GRPO due to its demonstrated effectiveness on LLMs. Extending MURPHY to other RLVR algorithms is conceptually straightforward.

4.1 MURPHY

In this work, we propose a method for leveraging execution feedback in language-model-driven code generation. We focus on a setting in which the model receives both *quantitative feedback*—for example, the proportion of test cases passed—and *qualitative feedback* that provides richer diagnostic

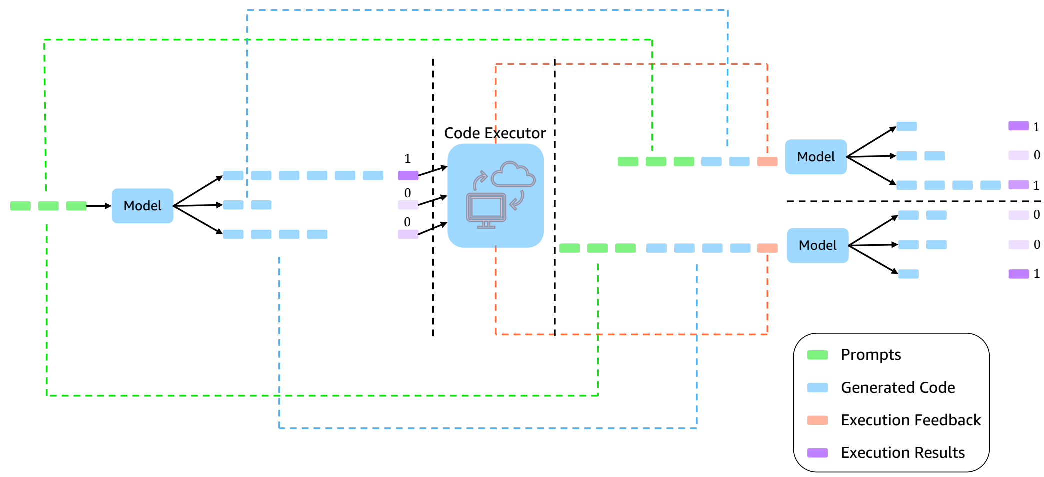

Figure 1: An illustration of our proposed approach, MURPHY. For a given prompt, G rollouts are generated and evaluated to obtain rewards. Rollouts that fail to achieve the maximum reward are augmented with executor feedback and re-prompted to produce an additional G rollouts, which are then evaluated. This process is repeated for a fixed number of turns. Rewards from the latter turns are propagated backward to earlier turns, and the GRPO objective is applied at each turn.

information about errors. A representative scenario arises in programming tasks with unit tests. In such tasks, the code generated by the language model can be executed against a predefined test suite. The execution results yield a quantitative measure, such as the number or proportion of tests passed, along with qualitative details indicating which specific test cases passed or failed. For each failing test case, we also provide execution traces, including error messages or stack traces, which can offer crucial insight for debugging. This combination of feedback types enables the model to refine its predictions with more targeted corrections, thereby improving the quality of convergence toward correct solutions. We incorporate both quantitative rewards and qualitative feedback into training by extending the GRPO framework.

For clarity of exposition, we describe both the forward and backward processes of our MURPHY algorithm. The forward process corresponds to tree construction or exploration, while the backward process performs credit reassignment and applies preference optimization to each subgroup at every level of the tree, favoring paths that lead to correct solutions.

FORWARD. In the first turn, the policy model receives the input prompts corresponding to the task and generates G candidate solutions per prompt. Each generation is evaluated on its associated test suite to compute: (i) a numerical reward, defined as the proportion of test cases passed, and (ii) qualitative feedback, consisting of identifiers for failed test cases and their execution outputs. Advantage scores are then computed by standardizing rewards within each group (i.e., generations sharing the same input prompt). This completes the first turn. In subsequent turns, for generations that did not achieve the maximum reward, the corresponding qualitative feedback is appended to the context, and the model is invoked to generate another G candidates. Rewards and advantages are computed as before. This process is repeated iteratively for a fixed number of turns.

BACKWARD. At this turn, we have accumulated advantage scores across turns. While some early generations may yield low rewards, appending execution feedback to the context can improve performance in later turns. Correct solutions may only appear after several refinement steps, making it essential to propagate credit to the intermediate generations that contribute to eventual success, rather than assigning reward solely based on immediate outcomes. We formalize this credit assignment problem and present the general update equation below, using notation consistent with Sec. 3, with additional definitions introduced here for clarity.

Let s denote the turn index, S the total number of turns, and G_s the number of generations per turn. We denote the model policy by $\pi(\cdot \mid \cdot)$, $\mathcal{P}(Q)$ the distribution over input prompts/questions Q, and O the output space. For the i-th generation in a group at turn s, let $o_{s,i} \in O$ denote the full generation trajectory, $o_{s,i,t}$ the t-th token, and $o_{s,i,<t}$ all tokens preceding t. The term $\hat{A}_{s,i,t}$ denotes the advantage of token t for the i-th generation at turn s. Let $c_{s,i}$ denote the feedback obtained at the end of turn s from the executor for generation i within a group along with the LLM generated code for that generation, together with an additional prompt that elicits model self-reflection. We define $q_{s-1}(i) := q_{s-1,\lfloor \frac{i-1}{G} \rfloor+1} := q,\ c_{s-1,\lfloor \frac{i-1}{G} \rfloor+1}$ as the prompt at turn s for generation i. Note that the G generations at turn s corresponding to generation $i-1$ are all conditioned on the same prompt. Finally, $D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{ref}})$ denotes the KL divergence between the current policy and a reference policy, computed over all tokens in the generated sequences.

Definition 2. (MURPHY Objective)

$$\mathcal{J}(\theta) = \mathbb{E}_{\substack{\{o_{s,i}\}_{s,i=1}^{S,G} \sim \pi_{\theta_{\text{old}}}(O|q) \\ q \sim \mathcal{P}(Q)}} \left[\sum_{s=1}^{S} \sum_{i=1}^{G_s} \frac{1}{G_s |o_{s,i}|} \sum_{t=1}^{|o_{s,i}|} \min\Bigg(\frac{\pi_\theta(o_{s,i,t} \mid q_{s-1}(i), o_{s,i,<t})}{\pi_{\theta_{\text{old}}}(o_{s,i,t} \mid q_{s-1}(i), o_{s,i,<t})} \hat{A}_{s,i,t}, \right.$$
$$\left. \text{clip}\left(\frac{\pi_\theta(o_{s,i,t} \mid q_{s-1}(i), o_{s,i,<t})}{\pi_{\theta_{\text{old}}}(o_{s,i,t} \mid q_{s-1}(i), o_{s,i,<t})}, 1-\epsilon, 1+\epsilon \right) \hat{A}_{s,i,t} \Bigg) \right] - \beta D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{ref}})$$

Note that since our approach extends GRPO to an online, feedback-driven refinement process over S turns, we have $G_s = G^s$. A natural question that arises is how to propagate rewards from later turns to earlier turns. To address this, we consider two possible design choices, which we describe below.

1. **Max-Reward:** For a given generation at turn s, each generation $o_{s,i}$ denotes an output that can be executed against test cases. We obtain G generations conditioned on $q_{s-1}(i)$. Let the rewards for turn $s-1$ (corresponding to generation $o_{s-1,i}$) be denoted as $r_{s-1,i}$. The rewards for the next G generations at turn s, conditioned on this output, the prompt, and the execution feedback (e.g., console logs or unit test results), are denoted as $r_{s,\,G(i-1)+1}, r_{s,\,G(i-1)+2}, \ldots, r_{s,\,Gi}$. We assign the reward for the previous turn by taking the maximum future reward and updating the current reward accordingly, but only if the previous turn does not already pass. Since our reward is defined as the proportion of test cases passed, we propagate the reward backwards as follows: $r_{s-1,i} = \max\big(r_{s-1,i}, \mathbf{1}(r_{s-1,i} \neq 1) \cdot \max_{j \in \{G(i-1)+1,\, G(i-1)+2,\, \ldots,\, Gi\}} r_{s,j}\big)$, where $\mathbf{1}(r_{s-1,i} \neq 1)$ is an indicator function that equals 1 if the prior turn failed and 0 otherwise, ensuring that future rewards are only propagated when the current turn does not already achieve a perfect score.
2. **Value Function:** This is similar to Bellman updates; however, we do not explicitly maintain a value function. Following the same notation as before, we compute the reward at turn i as $r_{s-1,i} = r_{s-1,i} + \gamma \cdot \frac{1}{G} \big[\sum_{j=1}^{G} r_{s,\,G\cdot(i-1)+j} \big]$, where γ denotes the discount factor.

We note that once the rewards are updated according to the chosen design, the advantage is computed in the same way as in standard GRPO. Specifically, conditioned on a prompt and for G generations, the mean is subtracted from each reward and the result is divided by the standard deviation.

4.2 Strategies for pruning rollouts in MURPHY

MURPHY due to multiple turns introduces significant computational cost. In particular, the number of rollouts grows exponentially with the number of turns: given a starting prompt (without feedback) at turn s, we obtain $G_s = G^s$ generations. While system optimizations such as vLLM with paged attention and KV caching [9] make the generation process relatively cheap, the optimization step remains computationally intensive. To address this challenge, we investigate several pruning strategies that reduce the number of rollouts at each turn, thereby making MURPHY computationally tractable without compromising performance. We describe these design strategies below.

1. **Max Variance Reward:** We derive inspiration from [18], where the authors show that pruning trajectories to retain those with maximum reward variance can reduce optimization cost while maintaining performance comparable to GRPO. Given a pruning budget B_s for an arbitrary turn s, we retain only B_s trajectories conditioned on the prompt at that turn. Let the function `max_variance`$\big(\mathbf{1}(r_{s-1,i} = 1) \cdot r_{s,\,G(i-1)+1},\ \mathbf{1}(r_{s-1,i} = 1) \cdot r_{s,\,G(i-1)+2},\ \ldots,\ \mathbf{1}(r_{s-1,i} = 1) \cdot r_{s,\,Gi},\ B_s\big)$ return the B_s trajectories, along with

their corresponding rewards and feedback, that exhibit the highest variance. Reward propagation is then performed as described in Subsec. 4.1, but only over this pruned subset. A key distinction from [18] is that in our approach, the maximum variance computation is performed only if the prior turn passes. Otherwise, the selected subset has zero reward due to masking, and B_s trajectories with their rewards are randomly sampled.

2. **Max Generation Batch Score:** Another pruning strategy is to optimize trajectories at the next turn (and propagate rewards backward) only for the top-performing batches when the G generations for a given prompt—conditioned on a generation $o_{s,i}$ for all $i \in G_s$—are ranked by score. We sort these batches by their scores and retain only the top B_s. For example, let $B_{s+1} = 1$. Consider turn s with $i \in \{1, 2\}$ and $G_{s+1} = 2$. In this case, $o_{s+1,1}$ and $o_{s+1,2}$ are generated from $o_{s,1}$ along with its corresponding prompt and execution feedback, while $o_{s+1,3}$ and $o_{s+1,4}$ are generated from $o_{s,2}$ and its corresponding execution feedback and prompt. We compute a batch score for $\{o_{s+1,1}, o_{s+1,2}\}$, denoted `score_1`, and another for $\{o_{s+1,3}, o_{s+1,4}\}$, denoted `score_2`. If `score_2` $>$ `score_1`, we retain $o_{s+1,3}$ and $o_{s+1,4}$ while discarding $o_{s+1,1}$ and $o_{s+1,2}$, assigning zero reward to the discarded generations. Note that there are multiple ways to assign a score to a batch of generations. Inspired by UCB sampling [1], we assign a score as $\alpha_1\mu + \alpha_2\sigma$, where μ and σ denote the mean and standard deviation of the rewards from the G generations conditioned on a given prompt at turn $s + 1$, and α_1 and α_2 are hyperparameters.

5 Experiments

In this section, we begin with an overview of the models and datasets used in our experiments. We then describe the experimental setup in detail and conclude with a discussion of the results.

Models: We evaluate our framework using two open-source model families: Qwen3 (1.7B, 4B) [20] and OLMo2 (OLMo-2-1124-7B-Instruct) [12]. Their diversity allows us to assess performance across both architectures and model sizes.

Datasets and Metrics:

Training Dataset: We train the models using 1,000 samples randomly drawn from the Kodcode dataset [19].

Evaluation Datasets: We evaluate the trained models on a suite of programming benchmarks covering coding and reasoning tasks: HumanEval [3], MBPP [2], and BigCodeBench-Hard [27]. For BigCodeBench-Hard, where visible unit tests are not provided, we randomly sample two test cases from the full test suite to construct visible tests.

Metrics and Evaluation Protocol: To analyze self-correction and reasoning refinement, we integrate the models into the Reflexion framework and measure *pass@1* under two settings:

1. *Single iteration*, equivalent to standard input–output prompting.
2. *Three iterations*, where feedback from visible test cases in earlier iterations is appended to the prompt for subsequent generations.

Iterations terminate once all visible test cases pass or the maximum iteration limit is reached. The final solution is then evaluated on hidden test cases, and *pass@1* is reported. Results are presented in Tab. 1.

Hyper-parameters: We set the KL regularization factor β to 0.04, the learning rate to 10^{-6}, and the weight decay to 0.1 for both GRPO and MURPHY. To save on computational cost, the number of stages in MURPHY is restricted to 2. For GRPO and the first stage of MURPHY, we use 8 rollouts per prompt, while the second stage of MURPHY uses up to 64 rollouts per prompt. To ensure a fair computational comparison, we also run GRPO with 72 rollouts for Qwen3-1.7B and OLMo2-7B-Instruct. We would like to note that all evaluations are performed with Temperature set to 0.6 and TopP set to 0.95.

5.1 MURPHY EXPERIMENTS

We test three models—Qwen3-1.7B [20], OLMo-2-1124-7B-Instruct [12], and Qwen3-4B—trained on 1,000 samples from the Kodcode dataset [19]. Their performance is assessed on the benchmark

Model	Rollouts	HumanEval (%)		MBPP (%)		BigCodeBench (%)	
		Iter-1	Iter-3	Iter-1	Iter-3	Iter-1	Iter-3
Qwen3-1.7B							
Base	–	74.19 ± 1.95	80.07 ± 1.27	43.47 ± 0.64	53.93 ± 3.52	7.20 ± 2.73	18.24 ± 1.79
GRPO	8	<u>77.85 ± 0.70</u>	83.94 ± 1.27	43.27 ± 0.23	57.07 ± 1.55	**10.81 ± 1.17**	18.92 ± 1.35
GRPO	72	77.42 ± 0.59	82.11 ± 1.76	<u>45.93 ± 1.60</u>	57.20 ± 1.40	10.58 ± 1.95	**20.49 ± 2.17**
MURPHY (No Fan out)	144	70.33 ± 0.93	82.11 ± 0.35	**46.00 ± 0.40**	<u>58.53 ± 1.03</u>	**13.29 ± 1.41**	18.24 ± 1.17
MURPHY- Max	72	**79.67 ± 3.01**	**86.58 ± 1.06**	44.73 ± 0.50	**62.00 ± 1.91**	6.98 ± 3.72	**20.25 ± 3.07**
Qwen3-4B							
Base	–	90.04 ± 3.13	93.49 ± 0.93	52.13 ± 0.42	70.93 ± 1.01	17.56 ± 1.78	36.03 ± 3.05
GRPO	8	88.61 ± 0.93	94.71 ± 0.93	51.73 ± 0.23	<u>72.87 ± 0.90</u>	20.95 ± 0.67	39.64 ± 2.73
MURPHY- Max	72	**92.48 ± 0.61**	**95.73 ± 0.31**	**53.33 ± 1.15**	**73.33 ± 1.31**	**22.52 ± 2.47**	**41.44 ± 1.09**
OLMo-2-1124-7B-Instruct							
Base	–	37.20 ± 0.86	46.04 ± 0.43	19.90 ± 0.42	29.60 ± 0.28	1.35 ± 0.00	3.72 ± 0.48
GRPO	8	<u>45.12 ± 0.00</u>	48.17 ± 1.06	28.53 ± 0.23	34.87 ± 1.75	**2.70 ± 0.00**	**6.31 ± 1.41**
GRPO	72	41.26 ± 0.35	43.70 ± 0.93	**32.87 ± 0.23**	35.80 ± 1.00	0.68 ± 0.00	2.70 ± 0.68
MURPHY (No Fan out)	144	39.02 ± 0.00	41.87 ± 0.93	28.33 ± 0.31	33.60 ± 0.69	<u>1.13 ± 1.41</u>	<u>2.25 ± 0.39</u>
MURPHY	72	**45.53 ± 0.70**	**52.24 ± 1.96**	<u>29.33 ± 0.90</u>	**39.67 ± 1.29**	1.80 ± 0.39	3.38 ± 1.17

Table 1: Performance of Qwen3-1.7B, Qwen3-4B, and OLMo-2-1124-7B-Instruct variants on HumanEval, MBPP, and BigcodeBench benchmarks, reported as pass@1 accuracy (% mean ± stdev). Best results are **bold**, second-best are <u>underlined</u>. Rollouts column denotes the total number of generations across both the stages. Note that the reported mean and standard deviations are computed over three independent evaluation runs.

datasets described in Sec. 5 using the Reflexion framework. We report *pass@1* results under both single-iteration and multi-iteration settings, as summarized in Tab. 1. We note that MURPHY-Max denotes MURPHY with the maximum-reward objective, while MURPHY (No Fan-Out) refers to a configuration where no fan-out is applied in the second stage: for each first-stage generation, we produce 72 candidates, and for each candidate, the model is prompted once with feedback to produce a single refinement. This ablation highlights both the importance of applying GRPO across stages and the benefits of fan-out in the second stage.

Reflexion: Single-iteration setting. (Corresponds to Iter-1 in Tab. 1) Models trained with the GRPO objective consistently outperform their base counterparts, with gains that are sometimes substantial (e.g., a $\sim 8\%$ lift on HumanEval for OLMo-2-1124-7B-Instruct). Increasing the number of GRPO rollouts does not yield further significant improvements in this setting. MURPHY achieves competitive or superior performance, with up to a 4% gain over GRPO. The most pronounced benefits of MURPHY, however, emerge in the multi-iteration setting.

Reflexion: Multiple-iteration setting. (Corresponds to Iter-3 in Tab. 1) We repeat the experiments with three iterations in the Reflexion framework to study the self-correction and reasoning-refinement capabilities of the models. Increasing the number of iterations yields significant performance improvements across all models and datasets. Models trained with MURPHY consistently outperform both GRPO-trained and base models, achieving gains of up to 5% over GRPO. These results highlight the benefits of multi-turn reflective optimization for enhancing self-correction and reasoning refinement.

5.2 Ablative Study-1: Max Reward vs Value Function

As described in Subsec. 4.1, we consider two strategies for propagating rewards from the next stage to the current stage: **Max Reward** and **Value Function**. As an ablation, we train Qwen3-1.7B and OLMo-2-1124-7B-Instruct on 1,000 samples from the Kodcode dataset using each strategy, and report the results in Tab. 2. Across both models and multiple iterations of Reflexion, the Max Reward strategy is consistently greater than or equal to the Value Function strategy, regardless of the discount factor γ. The reason lies in the nature of non-binary rewards. The Value Function averages over all generations, which weakens the learning signal when only a few outputs achieve high rewards. For instance, if one generation passes all test cases (reward 1.0) while the rest fail (reward

0), the average reward baseline becomes small, reducing the relative advantage of the successful trajectory and making it harder to separate signal from noise. By contrast, Max Reward propagates the strongest outcome directly, ensuring that rare but valuable high-reward trajectories dominate the update. Intuitively, solving all test cases correctly is far harder than solving only a few, and the Max Reward strategy preserves this distinction. This makes it particularly effective in multi-turn settings where rewards are sparse and high-quality generations are rare. In binary reward tasks, where all non-zero rewards are equivalent, the difference between the two strategies is naturally less pronounced.

Model & Strategy	**Rollouts**	**HumanEval (%)**		**MBPP (%)**		**BigCodeBench (%)**	
		Iter-1	**Iter-3**	**Iter-1**	**Iter-3**	**Iter-1**	**Iter-3**
Qwen3-1.7B							
Max Reward	8	**79.67 $\pm$ 3.01**	**86.58 $\pm$ 1.06**	44.73 $\pm$ 0.50	**62.00 $\pm$ 1.91**	6.98 $\pm$ 3.72	20.25 $\pm$ 3.07
Value Function ($\gamma = 0.9$)	8	78.66 $\pm$ 2.11	84.76 $\pm$ 1.22	**46.53 $\pm$ 0.81**	60.53 $\pm$ 1.79	**10.81 $\pm$ 0.00**	**22.52 $\pm$ 5.67**
Value Function ($\gamma = 1$)	8	78.46 $\pm$ 0.70	85.57 $\pm$ 0.35	45.07 $\pm$ 1.10	60.93 $\pm$ 1.29	9.46 $\pm$ 2.44	20.27 $\pm$ 1.35
OLMo-2-1124-7B-Instruct							
Max Reward	8	**45.53 $\pm$ 0.70**	**52.24 $\pm$ 1.96**	29.33 $\pm$ 0.90	**39.67 $\pm$ 1.29**	1.80 $\pm$ 0.39	**3.38 $\pm$ 1.17**
Value Function ($\gamma = 1$)	8	37.40 $\pm$ 0.35	41.87 $\pm$ 1.27	27.87 $\pm$ 0.12	34.40 $\pm$ 2.40	0.90 $\pm$ 0.39	1.13 $\pm$ 0.39

Table 2: Ablation study comparing **Max Reward** vs **Value Function** reward propagation strategies across Qwen3-1.7B and OLMo-2-1124-7B-Instruct. Results are reported as pass@1 accuracy (% mean $\pm$ stdev). Best results are **bold**. Note that the reported mean and standard deviations are computed over three independent evaluation runs.

5.3 Ablative Study-2: Max Variance Reward vs Max Generation Batch Score

In Section Subsec. 4.2, we introduced two pruning strategies: Max Variance and Max Generation Batch Score. Here, we compare these approaches and show that Max Generation Batch Score is a stronger alternative, capable of recovering the same performance as MURPHY without pruning. We conduct an ablation study using the Qwen3-1.7B model, and report the results in Table Tab. 3.

Model & Strategy	**Updates**	**HumanEval (%)**		**MBPP (%)**		**BigCodeBench (%)**	
		Iter-1	**Iter-3**	**Iter-1**	**Iter-3**	**Iter-1**	**Iter-3**
MURPHY (Max)	72	**79.67 $\pm$ 3.01**	**86.58 $\pm$ 1.06**	**44.74 $\pm$ 0.50**	**62.00 $\pm$ 1.90**	**6.97 $\pm$ 3.72**	**20.25 $\pm$ 3.07**
MURPHY (Max) – Max Variance	36	82.34 $\pm$ 1.03	84.28 $\pm$ 2.01	45.73 $\pm$ 0.50	59.87 $\pm$ 1.70	11.03 $\pm$ 2.81	21.17 $\pm$ 2.07
MURPHY (Max) – Max Generation	40	**77.43 $\pm$ 2.20**	**86.17 $\pm$ 0.70**	**44.53 $\pm$ 0.90**	**61.20 $\pm$ 1.39**	**10.58 $\pm$ 2.56**	**24.54 $\pm$ 2.82**

Table 3: Comparison of MURPHY and its pruned variants across HumanEval, MBPP, and BigCodeBench-Hard. Results are reported as pass@1 accuracy (% mean $\pm$ stdev). Note that the reported mean and standard deviations are computed over three independent evaluation runs.

6 Conclusion & Limitations

In this work, we introduced MURPHY, a multi-turn reflective optimization framework that extends RLVR algorithms (demonstrated with the GRPO instantiation) by incorporating iterative self-correction through both quantitative and qualitative execution feedback. By grounding optimization in intermediate signals and propagating rewards backward across refinement stages, MURPHY achieves consistent pass@1 improvements over standard GRPO across multiple LLM families and benchmark datasets, with the largest gains observed in multi-iteration scenarios where reasoning refinement is critical. These results highlight the importance of integrating structured feedback loops directly into the optimization process for code generation tasks. Our design, however, involves trade-offs. The multi-turn architecture increases computational cost, and although pruning strategies mitigate this overhead, MURPHY remains more resource-intensive than single-stage baselines. Moreover, our experiments were limited to structured feedback in code generation and restricted refinement depth to two stages for tractability, leaving open questions about generalization to less structured or noisier feedback, deeper refinement chains, and broader measures of agentic performance such as robustness or alignment with developer intent. For future work, we plan to explore more efficient rollout selection mechanisms, potentially via learned reward models, to reduce overhead without sacrificing accuracy. We also aim to test MURPHY in more diverse domains, such as multi-step scientific reasoning or interactive debugging, where feedback is less deterministic and more varied. Another promising direction is enabling dynamic stage counts that adapt to task complexity or confidence signals, as well as combining MURPHY with search-based inference-time strategies to jointly enhance training and inference-time reasoning refinement. Together, these extensions could improve the applicability, efficiency, and impact of MURPHY across a broader range of agentic reasoning challenges.

References

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[4] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

[5] Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding code LLMs in execution feedback with reinforcement learning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=PzSG5nKe1q.

[6] Arnav Kumar Jain, Gonzalo Gonzalez-Pumariega, Wayne Chen, Alexander M Rush, Wenting Zhao, and Sanjiban Choudhury. Multi-turn code generation through single-step rewards. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=aJeLhLcsh0.

[7] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. code$_{g}en_{l}lm_{s}urvey$. *ACM Trans. Softw. Eng. Methodol.*, $July 2025. ISSN 1049-331X. doi:.$ URL https://doi.org/10.1145/3747588.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning, 2024. URL https://arxiv.org/abs/2409.12917.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Vijay Lingam, Behrooz Omidvar Tehrani, Sujay Sanghavi, Gaurav Gupta, Sayan Ghosh, Linbo Liu, Jun Huan, and Anoop Deoras. Enhancing language model agents using diversity of thoughts. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ZsP3YbYeE9.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. SWE-lancer: Can frontier LLMs earn $1 million from real-world freelance software engineering? In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=xZXhFg43EI.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 olmo 2 furious, 2025. URL https://arxiv.org/abs/2501.00656.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haotian Yao, Haotian Zhao, Haoyu Lu, Haoze Li, Haozhen Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia Chen, Jianhang Guo, Jianlin Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Lidong Shi, Ling Ye, Longhui Yu, Mengnan Dong, Neo Zhang, Ningchen Ma, Qiwei Pan, Qucheng Gong, Shaowei Liu, Shengling Ma, Shupeng Wei, Sihan Cao, Siying Huang, Tao Jiang, Weihao Gao, Weimin Xiong, Weiran He, Weixiao Huang, Weixin Xu, Wenhao Wu, Wenyang He, Xianghui Wei, Xianqing Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xuehai Pan, Y. Charles, Yang Li, Yangyang Hu, Yangyang Liu, Yanru Chen, Yejie Wang, Yibo Liu, Yidao Qin, Yifeng Liu, Ying Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang, Zhaowei Li, Zhen Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Ziyao Xu, Zonghan Yang, and Zongyu Lin. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying llm-based software engineering agents. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. 10.1145/3715754. URL https://doi.org/10.1145/3715754.

Yixuan Even Xu, Yash Savani, Fei Fang, and Zico Kolter. Not all rollouts are useful: Down-sampling rollouts in llm reinforcement learning. *arXiv preprint arXiv:2504.13818*, 2025.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. *arXiv preprint arXiv:2503.02951*, 2025.

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, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering.

In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=mXpq6ut8J3.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, 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, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Yufeng Yuan, Yu Yue, Ruofei Zhu, Tiantian Fan, and Lin Yan. What's behind ppo's collapse in long-cot? value optimization holds the secret, 2025. URL https://arxiv.org/abs/2503.01491.

Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, Xiangpeng Wei, Xiangyu Yu, Gaohong Liu, Juncai Liu, Lingjun Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Ru Zhang, Xin Liu, Mingxuan Wang, Yonghui Wu, and Lin Yan. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks, 2025. URL https://arxiv.org/abs/2504.05118.

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

Li Zhong, Zilong Wang, and Jingbo Shang. Debug like a human: A large language model debugger via verifying runtime execution step by step. In *Findings of the Association for Computational Linguistics ACL 2024*, pp. 851–870, 2024.

Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=YrycTjllL0.