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. We introduce MURPHY, a multi-turn RLVR framework that incorporates execution feedback directly into training, extending GRPO to optimize over multi-turn trajectories where models iteratively refine solutions. MURPHY combines a feedback-conditioned rollout tree with trajectory-level credit assignment, and uses pruning to reduce the cost of multi-turn optimization. Evaluations on code generation benchmarks with two model families show that MURPHY consistently improves multi-iteration performance, achieving up to an 8% absolute gain in pass@1 over compute-matched GRPO baselines, and outperforming the prior leading method that incorporates multi-turn execution feedback.

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*

A growing body of work explores large language models (LLMs) as software engineering agents that interact with their environment through code execution and feedback (Shinn et al., 2023; Lingam et al., 2025; Miserendino et al., 2025). Rather than producing a single static response, these systems operate within agentic scaffolds (Yao et al., 2023; Zhou et al., 2024) that guide iterative reasoning and allow LLMs to act, observe, and improve over multiple rounds of interaction. For example, in a typical coding agentic scaffold (Shinn et al., 2023; Lingam et al., 2025), the agent generates a solution by performing a single or series of actions, and executes it for evaluation, often through unit tests or other automated checks. When execution fails, the agent receives feedback such as error messages, stack traces, or failing test cases, and is re-prompted with the original task, its previous attempt, and the new feedback. This process continues for multiple turns until the model succeeds or reaches a fixed iteration limit. These systems highlight the growing ability of LLMs to reason and adapt through environmental feedback at inference time. In code generation tasks (Jiang et al., 2025), such feedback naturally arises from execution logs, compiler errors, or test results. However, these methods (Shinn et al., 2023; Lingam et al., 2025) remain fundamentally *training-free*: they improve model behavior through structured inference via textual feedback rather than through parameter updates.

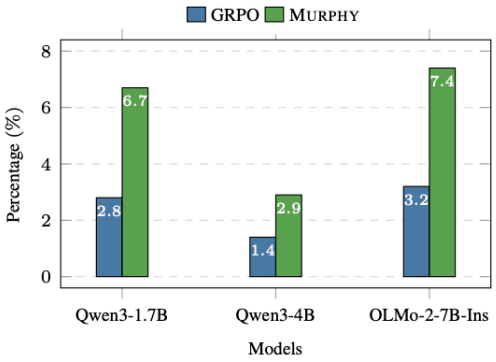

Figure 1. Percentage change in coding problems solved by models trained with MURPHY and GRPO over the base model across three models and datasets. For Qwen3-4B, we additionally include results on the Aider Polyglot benchmark. MURPHY-trained models solve up to 4.2% more problems than GRPO. See Tab. 1 and Subsec. 5.1 for details.

On the other hand, training methodologies such as Reinforcement Learning with Verifiable Rewards (RLVR) have enabled a new generation of language models (OpenAI et al., 2024; DeepSeek-AI et al., 2025; Team et al., 2025; Yang et al., 2025) to exhibit strong reasoning capabilities across mathematics, coding, and general problem-solving. Recent RLVR algorithms, including Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and its extensions (Yue et al., 2025; Yu et al., 2025; Xu et al., 2025a), have become dominant approaches for post-training LLMs on verifiable reasoning tasks. However, GRPO and its extensions are fundamentally designed for a single-turn setting: they optimize model behavior using an isolated prompt–response–reward tuple, with no notion of multi-turn interaction defined in their objective. Here, a turn denotes one cycle in which the model receives a prompt (which includes feedback from its previous response), reasons based on the prompt, and generates a revised output. Taken together, these limitations underscore a methodological gap: *while inference-time agentic frameworks exploit iterative feedback to refine reasoning, GRPO optimizes solely from terminal rewards on a given prompt, without leveraging intermediate environmental feedback*. This motivates the following key research question.

Key Research Question: How can RLVR algorithms be extended to integrate iterative feedback, and to what extent does this improve reasoning and performance within agentic frameworks?

To address this question, we propose MURPHY, a novel RLVR algorithm that extends GRPO to a multi-turn setting by conditioning optimization on intermediate environmental feedback. Extending GRPO beyond a single turn is non-trivial: it requires defining how rewards obtained in later turns should be propagated backward to earlier attempts, so that intermediate reasoning and output turns that initially failed but ultimately led to success through feedback receive appropriate credit. At the first turn, MURPHY generates G_1 generations per prompt and computes group-based rewards as in GRPO. In subsequent turns, generations that fail to achieve the maximum reward, e.g., those failing unit tests or producing incorrect outputs, are refined using signals from the environment such as executor logs or test results. This feedback is appended to the original prompt, and the model is re-prompted to generate a new batch of G_s generations (where s denotes the turn) conditioned on the combined context (prompt, previous output, and feedback), repeating this process for a fixed number of turns. Rewards from successful final-turn rollouts are then propagated backward to earlier turns using MURPHY's credit-assignment criterion, allowing partially correct but improving attempts to receive credit. To manage the computational cost of multi-turn updates, MURPHY employs pruning strategies that retain only promising trajectories while bounding total gradient updates per turn. In summary, our main contributions are:

Main Contributions.

1. We introduce MURPHY, a multi-turn RLVR algorithm that extends GRPO to leverage execution feedback for grounded reasoning and self-correction in code generation tasks. (Sec. 4)
2. We design pruning mechanisms that enable scalable multi-turn RLVR training (MURPHY) while preserving performance. (Subsec. 4.1)
3. We evaluate MURPHY on three code generation benchmarks across two model families (OLMo, Qwen) and sizes (1.7B–7B), achieving up to 8% absolute pass@1 gains over GRPO baselines, while also outperforming μCode, a leading multi-turn execution-feedback method. (Sec. 5)

This work focuses on code generation tasks where execution provides structured, verifiable feedback, enabling us to measure feedback-driven refinement using agentic scaffolds at evaluation time.

2. Related Work

LLM Agents for Software Development. Recent studies (Jiang et al., 2025; Zhong et al., 2024) investigate LLM agents for code generation, bug fixing, and code migration. A central factor behind their progress is inference-time iterative frameworks (Shinn et al., 2023; Lingam et al., 2025), which leverage execution feedback and self-reflection to refine candidate programs (Yang et al., 2024; Xia et al., 2025). While such methods enhance inference pipelines, they leave the base model unchanged. In contrast, our work improves the model itself through training-time optimization, strengthening the reasoning and self-correction abilities that agentic frameworks depend on.

RLVR for LLM Reasoning. RL has emerged as a powerful paradigm for aligning LLMs with verifiable objectives. GRPO (Shao et al., 2024) renewed interest in RL as an efficient alternative to PPO (Schulman et al., 2017), achieving comparable reasoning performance with lower computational cost. Follow-up variants (Yue et al., 2025; Yu et al., 2025; Yuan et al., 2025; Zheng et al., 2025) improve stability, convergence, or shift optimization from token-level to sequence-level, yet they remain tailored to single-turn tasks. Our work is closely related to μCode (Jain

et al., 2025) and RLEF (Gehring et al., 2025), which incorporate execution feedback during training. μCode jointly trains a generator with a learned verifier that scores multi-turn code solutions, whereas RLEF applies PPO grounded in execution results. However, both rely on auxiliary value functions or verifier LLMs, increasing computational and data costs. In contrast, MURPHY extends GRPO to the multi-turn setting, achieving comparable grounding in execution feedback while retaining simplicity, efficiency, and architectural minimalism. See App. A for extended related work.

3. Background: GRPO

Group Relative Policy Optimization (GRPO; (Shao et al., 2024)) is a variant of Proximal Policy Optimization (PPO) designed to improve the efficiency and stability of policy updates in LLM fine-tuning. Unlike PPO, which estimates advantages using a learned value function (critic), GRPO replaces the critic with an empirical baseline: the mean reward across all generations produced by the model for the same prompt. In both methods, rewards are provided by a reward model. Specifically, for each input prompt, the model samples a set of G candidate responses, forming a *response group*. The reward model assigns a score to each response, and advantages are computed by standardizing rewards within the group: subtracting the group mean and dividing by the group standard deviation. This yields relative, normalized advantage values. As in PPO, GRPO may incorporate a penalty term to prevent the updated policy from drifting too far from the reference policy, typically enforced via a Kullback–Leibler (KL) divergence regularizer to ensure stable updates. A formal definition of the GRPO objective, along with additional details, is provided in App. C.

4. Proposed Method: MURPHY

Extending GRPO to multi-turn interaction requires formalizing how environmental feedback informs optimization. We study a feedback-rich code generation setting where model outputs are executed and scored against test suites, yielding two feedback types: (1) quantitative (e.g., proportion of test cases passed) and (2) qualitative (e.g., error traces or failing cases). MURPHY extends GRPO by introducing (i) a multi-turn rollout mechanism that conditions generation on feedback, and (ii) a credit-assignment scheme that propagates rewards from future successful turns to earlier attempts.

Multi-turn rollout: For each task prompt, the policy begins at turn 1 by generating G_1 candidate solutions, each evaluated for reward against the prompt's test suite. Failed generations are paired with their corresponding feedback and, together with the original prompt and prior outputs, form new conditioning contexts for subsequent turns. At turn s, the model generates G_s new candidates for each failed case, continuing this feedback–refinement process for a fixed number of turns. This per-prompt, iterative rollout enables progressive improvement conditioned on feedback. Unlike standard GRPO, MURPHY *delays advantage computation until rollouts in all turns are generated and processed*, allowing final rewards to retroactively shape credit assignment across prior turns.

Credit assignment: Once rewards for all turns are computed, MURPHY backpropagates rewards from the final turn to earlier turns using a temporal credit-assignment scheme. Although early generations may achieve low initial rewards, incorporating execution feedback in later turns often leads to successful refinements. To ensure that these intermediate steps are properly credited, rewards from later successful generations are distributed to earlier ones according to their contribution to eventual success. After reward redistribution, advantages are computed for each generation, and the MURPHY loss is applied to update the policy. Together, these mechanisms extend GRPO to handle multi-turn, feedback-conditioned optimization. Below, we formalize the *multi-turn rollout mechanism* and *credit-assignment* process and define the MURPHY objective, extending GRPO to incorporate multi-turn feedback.

Notation and formalism. We denote the current and old policy models by $\pi_\theta(\cdot \mid \cdot)$ and $\pi_{\theta_{\text{old}}}(\cdot \mid \cdot)$. π_{ref} denotes the reference model (i.e., the model prior to fine-tuning), which is kept fixed (not updated) throughout training. $\mathcal{P}(Q)$ denotes the distribution over input prompts/questions Q, and O the output space. As described earlier, in the first turn, the model generates G_1 candidate solutions for each prompt. For generations that do not attain the max reward, feedback is obtained from the environment. The model is then re-prompted with the original prompt, its previous output, and the corresponding feedback to produce G_s new generations per prompt at turn s. This iterative procedure naturally forms a tree structure, where the root corresponds to the original prompt, and each subsequent generation (augmented with feedback) forms a child node, with generations at turn $s+1$ linked to their parent at turn s (see Fig. 2).

Multi-turn rollout formalism. We define a feedback-conditioned rollout tree that captures how generations evolve over S turns. Let s denote the turn index and G_s the number of generations per prompt at turn s. We use $i_{[1:s]} = (i_1, \ldots, i_s)$ to index a path in the rollout tree, where i_j denotes the branch taken at turn j.

Turn 1: The model receives $q_{(1)} \sim \mathcal{P}(Q)$ and generates

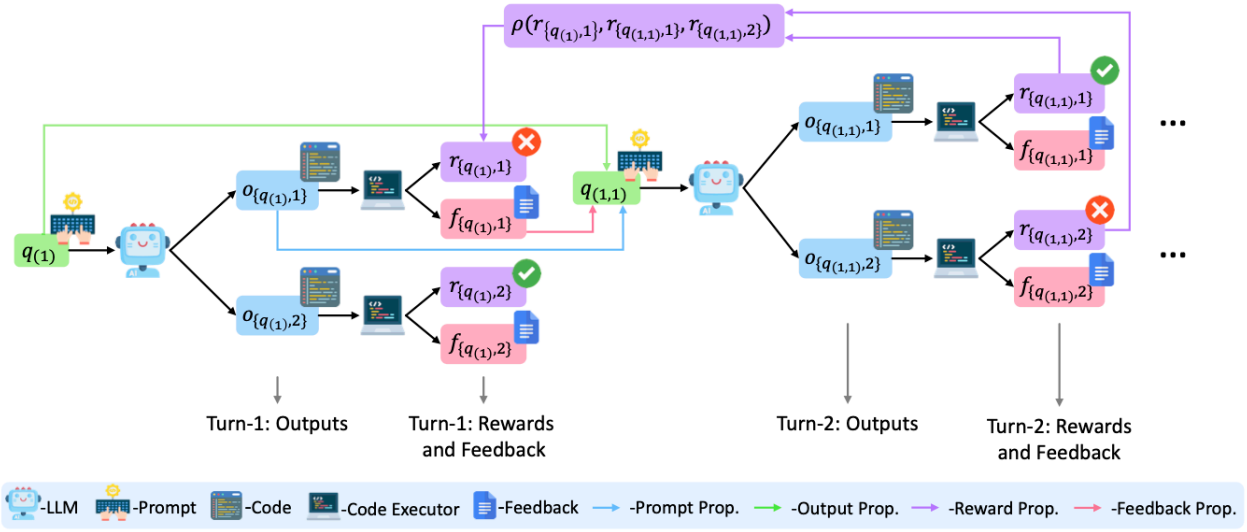

Figure 2. Overview of MURPHY. Given an input prompt ($q_{(\cdot)}$), G_1 code generations ($o_{(\cdot)}$) are generated and evaluated using a reward function ($r_{(\cdot)}$). Generations that do not achieve the maximum reward are revised based on executor feedback (f), combining the original prompt with the failed output, and re-prompted to generate another G_2 candidates. This iterative process continues for a fixed number of turns, with rewards from later turns propagated backward. The example illustrates the case with $G_1 = G_2 = 2$, where G_1, G_2 represents the number of rollouts per prompt, per turn, and $\rho(\cdot)$ denotes the credit assignment strategy.

a response group $\{o_{\{q_{(1)},j\}}\}_{j=1}^{G_1} \sim \pi_{\theta_{\text{old}}}(\cdot \mid q_{(1)})$. Each generation is executed to obtain a reward and feedback $(r_{\{q_{(1)},j\}}, f_{\{q_{(1)},j\}})$, where r denotes the proportion of tests passed and f contains qualitative executor feedback (e.g., failing tests, error messages).

Turn s+1: For any unsolved node at turn s (i.e., one that does not achieve the maximum reward), we form a feedback-conditioned prompt by concatenation: $q_{(s+1,i_{[1:s]})} = [q_{(s,i_{[1:s-1]})}, o_{\{q_{(s,i_{[1:s-1]})},i_s\}}, f_{\{q_{(s,i_{[1:s-1]})},i_s\}}]$, and sample G_{s+1} children

$$\{o_{\{q_{(s+1,i_{[1:s]})},k\}}\}_{k=1}^{G_{s+1}} \sim \pi_{\theta_{\text{old}}}(\cdot \mid q_{(s+1,i_{[1:s]})}),$$

each evaluated to obtain $(r_{\{q_{(s+1,i_{[1:s]})},k\}}, f_{\{q_{(s+1,i_{[1:s]})},k\}})$. This defines a rollout tree rooted at $q_{(1)}$ (see Fig. 2).

A complete path from the root to a leaf at turn S can be written as

$$q_{(1)} \to o_{\{q_{(1)},i_1\}} \to q_{(2,i_{[1:1]})} \to \cdots \to o_{\{q_{(S,i_{[1:S-1]})},i_S\}}.$$

Leaf nodes at turn S represent the final generations after refinement. We provide a more detailed formal treatment (including explicit Turn 2 expansion and tree construction details) in App. B.

Credit assignment formalism. After all outputs and corresponding rewards are generated and the rollout tree is constructed, we focus on assigning credit from later turns back to earlier ones. To achieve this, we explore two distinct strategies.

*Max Reward Strategy (*MARS*):* The Max Reward Strategy is defined recursively, proceeding from the final turn back to the root. Since the final turn S has no children, the rewards at this turn remain unchanged. We then consider turn $s = S - 1$. Let $o_{\{q_{(s,i_{[1:s-1]})},i_s\}}$ denote a generation at turn $s = S - 1$ with an associated reward $r_{\{q_{(s,i_{[1:s-1]})},i_s\}}$. If this generation already achieves the maximum reward, it has no children; otherwise, its children are defined as:

$$C\left(o_{\{q_{(s,i_{[1:s-1]})},i_s\}}\right) = \{o_{\{q_{(s+1,i_{[1:s]})},1\}}, \ldots, o_{\{q_{(s+1,i_{[1:s]})},G_S\}}\}$$

The corresponding set of rewards are defined as

$$C_r\left(o_{\{q_{(s,i_{[1:s-1]})},i_s\}}\right) = \{r_{\{q_{(s+1,i_{[1:s]})},1\}}, \ldots, r_{\{q_{(s+1,i_{[1:s]})},G_S\}}\}$$

which defaults to zero if there are no children. We then update the reward as:

$$r_{\{q_{(s,i_{[1:s-1]})},i_s\}} = \max\left(r_{\{q_{(s,i_{[1:s-1]})},i_s\}}, \max\left(C_r(o_{\{q_{(s,i_{[1:s-1]})},i_s\}})\right)\right)$$

This strategy assigns each node the maximum of its own reward and the best reward among its descendants. Intuitively, the descendant maximum represents the best

outcome achievable through refinement, while taking the outer maximum ensures a node's credit never decreases; even when feedback fails to improve performance. This formulation captures the maximum progress achievable from any refinement path starting at that node. The procedure operates recursively in a backward pass: rewards are first updated for all nodes at turn $S-1$ based on their children's values at turn S. This process continues backward through the tree, with each turn s receiving updated rewards based on the values from turn $s+1$, until reaching the root.

*Mean Reward Strategy (*MERS*):* The Mean Reward Strategy follows the same recursive credit assignment structure as the Max Reward Strategy (MARS), but differs in how rewards are propagated. Inspired by the return computation in REINFORCE (Williams, 1992), MERS updates each node's reward by incorporating the discounted *mean* of its children's rewards.

For a generation $o_{\{q_{(s,i_{[1:s-1]})},i_s\}}$ at turn s, let $\mathbb{I}_{\text{unsolved}}$ be an indicator that equals 1 if the problem remains unsolved at this node, and 0 otherwise. The update rule is:

$$r_{\{q_{(s,i_{[1:s-1]})},i_s\}} = \frac{r_{\{q_{(s,i_{[1:s-1]})},i_s\}} + \gamma \cdot \overline{C}_r\left(o_{\{q_{(s,i_{[1:s-1]})},i_s\}}\right)}{\mathbb{I}_{\text{unsolved}} \cdot (S-s)+1}$$

where $\gamma \in [0,1]$ is a discount factor controlling the influence of descendant rewards, and $\overline{C}_r(\cdot)$ denotes the mean reward over children of unsolved nodes (children of solved nodes are masked out since they represent terminal states).

The denominator equals $S-s+1$ for unsolved nodes and 1 for solved nodes. This depth-based normalization serves two purposes: (1) it prevents rewards from growing unboundedly during backward propagation, as unsolved nodes can potentially accumulate discounted contributions from up to $S-s$ future turns; and (2) it ensures fair comparison between nodes that solve at different turns, a node that solves immediately at turn s retains its full reward, while a node that fails but has successful descendants receives appropriately scaled credit that accounts for the additional refinement steps required. All other aspects of the recursive procedure remain identical to MARS.

MARS *vs* MERS: MARS propagates the maximum descendant reward to the earlier turns, emphasizing peak performance, whereas MERS propagates the mean reward, emphasizing stability and overall consistency. MARS captures best-case improvement, while MERS provides a smoother estimate of expected progress. Together, they offer complementary views of feedback-driven credit assignment.

MURPHY *Objective*: Once the rewards are reassigned according to the chosen credit assignment strategy (MARS or MERS), advantages are computed similar to standard GRPO. The adjusted rewards serve as the basis for computing normalized advantages at each turn. Conditioned on a prompt $\tilde{q}$ and for G_s generations, we normalize each reward by subtracting the mean reward and dividing by the standard deviation of the rewards obtained across the G_s generations, yielding the normalized advantage $\hat{A}^{\text{MURPHY}}_{\tilde{q},i,t}$. Additionally, as defined earlier, each i-th generation at turn s, $o_{\{q_{(s,i_{[1:s-1]})},i\}} \in O$, corresponds to a complete output trajectory, where $o_{\{q_{(s,i_{[1:s-1]})},i,t\}}$ denotes the t-th token and $o_{\{q_{(s,i_{[1:s-1]})},i,<t\}}$ the prefix up to (but excluding) token t. We denote the sequence length of the i-th generation by $|o_{\{q_{(s,i_{[1:s-1]})},i\}}|$. At each turn, the GRPO objective is applied using these credit-adjusted advantages. This per-turn optimization allows MURPHY to incorporate feedback from later turns into earlier updates, effectively extending GRPO to a multi-turn setting. Finally, the divergence between the current and reference policies is captured by $D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})$, computed over all tokens in the generated sequences. The resulting optimization objective, which integrates credit-assigned rewards, normalized advantages, and KL regularization at each turn, defines the MURPHY objective, distinguishing it from the standard GRPO formulation. The full MURPHY objective is presented in Def. 1.

Definition 1. (MURPHY Objective)

$$\mathcal{J}_{\text{MURPHY}}(\theta) = \mathbb{E}_{q\sim\mathcal{P}(Q)}\Big[\sum_{s,i_1,\ldots i_S} \mathcal{J}_{q_{(s,i_{[1:s]})}}(\theta)\Big]$$

Where the per prompt objective at each turn s is:

$$\mathcal{J}_{q_{(s,i_{[1:s-1]})}}(\theta) = \mathbb{E}_{\{o_{(\tilde{q},i)}\}_{i=1}^{G_s}\sim\pi_{\theta_{\text{old}}}(O|\tilde{q})}\Bigg[\sum_{i=1}^{G_s} \frac{1}{G_s|o_{\{\tilde{q},i\}}|} \sum_{t=1}^{|o_{\{\tilde{q},i\}}|} \Bigg(\min\Big(R_\theta(\tilde{q},i,t)\hat{A}^{\text{MURPHY}}_{\tilde{q},i,t}, \text{clip}\left(R_\theta(\tilde{q},i,t),\,1-\epsilon,\,1+\epsilon\right)\cdot \hat{A}^{\text{MURPHY}}_{\tilde{q},i,t}\Big) - \beta D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})\Bigg)\Bigg]$$

with, $\tilde{q} = q_{(s,i_{[1:s-1]})}$

$$R_\theta(\tilde{q},i,t) = \frac{\pi_\theta(o_{\{\tilde{q},i,t\}} \mid \tilde{q}, o_{\{\tilde{q},i,<t\}})}{\pi_{\theta_{\text{old}}}(o_{\{\tilde{q},i,t\}} \mid \tilde{q}, o_{\{\tilde{q},i,<t\}})}$$

And, $\hat{A}^{\text{MURPHY}}_{\tilde{q},i,t}$ denotes the advantage.

Note. The design of MURPHY is broadly applicable across a range of RLVR algorithms, including PPO (Schulman et al., 2017) and various extensions of GRPO (Ahmadian et al., 2024; Yu et al., 2025; Yue et al., 2025). In this work, we focus on GRPO due to its strong empirical performance in aligning LLMs (DeepSeek-AI et al., 2025; Yang et al., 2025). Extending MURPHY to other RLVR variants is conceptually straightforward, as it builds on the same underlying principles. While our experiments primarily focus on code generation, where rich, verifiable feedback is readily available, the framework can naturally extend to other domains such as mathematics or logical reasoning, provided suitable forms of feedback are accessible.

4.1. Pruning Strategies in MURPHY

By default, MURPHY sets $G_s = G$ for all turns, maintaining a fixed number of generations per prompt throughout the multi-turn process. However, this multi-turn setup introduces significant computational cost. In the worst case, when success is achieved only at the final turn S, the number of generations per prompt can grow exponentially to G^S, resulting in a large search tree and substantial memory overhead. This makes MURPHYcomputationally expensive to optimize. While system-level optimizations such as vLLM with paged attention and KV caching (Kwon et al., 2023) make the generation process relatively efficient, the optimization phase remains costly. Large-scale rollouts can quickly exhaust GPU memory, and batching schemes that treat each generated sample as an independent training batch tend to be prohibitively slow. To address these challenges, we introduce two *pruning strategies* that reduce the number of rollouts at each turn, thereby making MURPHY computationally tractable without compromising performance. We describe these strategies in detail below.

*Intra-Group Pruning (*INTRAP*)*: In INTRAP, pruning operates recursively: starting from the children of level $S-1$, each pruning step is followed by reward propagation as described in Sec. 4, and the process continues backward until reaching the root. This ordering, pruning before reward reassignment, ensures that only informative trajectories are retained for credit propagation across turns. At each turn s, given a pruning budget b, we retain only the b trajectories conditioned on the prompt whose rewards contribute most to the total reward variance within that group. The remaining trajectories, along with all their descendants, are discarded, and the process proceeds recursively until the root is reached. This approach is inspired by Xu et al. (2025a); they demonstrate that retaining trajectories with the highest reward variance within a group can reduce optimization cost while maintaining performance comparable to GRPO. We extend this principle to the multi-turn setting of MURPHY.

*Inter-Group Pruning (*INTERP*)*: In INTERP, the goal is to prune entire groups of children corresponding to generations conditioned on a given prompt. Specifically, each generation at a turn has a group of children, and we decide how many of these groups to retain based on a pruning budget b. To rank the groups, we assign a score inspired by the UCB sampling strategy (Auer et al., 2002), defined as $\alpha_1\mu + \alpha_2\sigma$, where μ and σ denote the mean and standard deviation of the rewards within each group of children. The groups are ranked in descending order of this score, and only the top b groups are retained, with b determining the available computational budget. The remaining groups and their descendants are discarded. Similar to IntraP, this process is applied recursively and proceeds backward from the latter turns to the earlier ones. At each level, pruning is performed first, followed by credit assignment. In our experiments, we set $\alpha_1 = 0$ and $\alpha_2 = 1$, which effectively prioritizes groups exhibiting higher reward variance. Intuitively, a higher standard deviation not only captures greater variability within a group but also indicates that the model is uncertain about that group's performance. Such groups likely contain some generations with near-maximum rewards, making them valuable targets for further optimization. In contrast, groups where all rewards are uniformly high (near 1) or uniformly low (near 0) provide limited learning signal, as the model either has already mastered or completely failed the underlying behavior. High-variance groups, therefore, represent the most informative regions for continued improvement.

5. Experiments

6. Conclusion & Limitations

We introduced MURPHY, a multi-turn reflective reinforcement learning framework that extends RLVR algorithms by incorporating iterative self-correction through both quantitative and qualitative feedback. By grounding optimization in intermediate feedback signals and propagating rewards across refinement turns, MURPHY consistently improves reasoning and code generation performance, particularly in multi-iteration settings where feedback-driven refinement is crucial. These findings underscore the value of integrating structured feedback directly into the optimization process. While effective, MURPHY's multi-turn design increases computational cost; pruning partially mitigates this but it remains more resource-intensive than single-turn baselines.

We study structured execution feedback for code, and generalization to noisier feedback, deeper refinement, and broader agentic objectives is open. In future work, we aim to develop adaptive turn/rollout selection and tool-augmented optimization (retrieval, execution, APIs).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Ahmadian, A., Cremer, C., Gallé, M., Fadaee, M., Kreutzer, J., Pietquin, O., Üstün, A., and Hooker, S. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267. Association for Computational Linguistics, August 2024. doi: 10.18653/v1/2024.acl-long.662. URL https://aclanthology.org/2024.acl-long.662/.

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Chen, M., Sun, L., Li, T., Sun, H., Zhou, Y., Zhu, C., Wang, H., Pan, J. Z., Zhang, W., Chen, H., Yang, F., Zhou, Z., and Chen, W. Research: Learning to reason with search for llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2503.19470.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Gehring, J., Zheng, K., Copet, J., Mella, V., Cohen, T., and Synnaeve, G. 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.

Jain, A. K., Gonzalez-Pumariega, G., Chen, W., Rush, A. M., Zhao, W., and Choudhury, S. Multi-turn code generation through single-step rewards. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=aJeLhLcsh0.

Jiang, J., Wang, F., Shen, J., Kim, S., and Kim, S. A survey on large language models for code generation. *ACM Trans. Softw. Eng. Methodol.*, July 2025. ISSN 1049-331X. doi: 10.1145/3747588. URL https://doi.org/10.1145/3747588.

Jin, B., Zeng, H., Yue, Z., Yoon, J., Arik, S. O., Wang, D., Zamani, H., and Han, J. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient

memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Lingam, V., Tehrani, B. O., Sanghavi, S., Gupta, G., Ghosh, S., Liu, L., Huan, J., and Deoras, A. Enhancing language model agents using diversity of thoughts. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ZsP3YbYeE9.

Miserendino, S., Wang, M., Patwardhan, T., and Heidecke, J. 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.

OLMo, T., Walsh, P., Soldaini, L., Groeneveld, D., Lo, K., Arora, S., Bhagia, A., Gu, Y., Huang, S., Jordan, M., Lambert, N., Schwenk, D., Tafjord, O., Anderson, T., Atkinson, D., Brahman, F., Clark, C., Dasigi, P., Dziri, N., Guerquin, M., Ivison, H., Koh, P. W., Liu, J., Malik, S., Merrill, W., Miranda, L. J. V., Morrison, J., Murray, T., Nam, C., Pyatkin, V., Rangapur, A., Schmitz, M., Skjonsberg, S., Wadden, D., Wilhelm, C., Wilson, M., Zettlemoyer, L., Farhadi, A., Smith, N. A., and Hajishirzi, H. 2 olmo 2 furious, 2025. URL https://arxiv.org/abs/2501.00656.

OpenAI, :, Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., Iftimie, A., Karpenko, A., Passos, A. T., Neitz, A., Prokofiev, A., Wei, A., Tam, A., Bennett, A., Kumar, A., Saraiva, A., Vallone, A., Duberstein, A., Kondrich, A., Mishchenko, A., Applebaum, A., Jiang, A., Nair, A., Zoph, B., Ghorbani, B., Rossen, B., Sokolowsky, B., Barak, B., McGrew, B., Minaiev, B., Hao, B., Baker, B., Houghton, B., McKinzie, B., Eastman, B., Lugaresi, C., Bassin, C., Hudson, C., Li, C. M., de Bourcy, C., Voss, C., Shen, C., Zhang, C., Koch, C., Orsinger, C., Hesse, C., Fischer, C., Chan, C., Roberts, D., Kappler, D., Levy, D., Selsam, D., Dohan, D., Farhi, D., Mely, D., Robinson, D., Tsipras, D., Li, D., Oprica, D., Freeman, E., Zhang, E., Wong, E., Proehl, E., Cheung, E., Mitchell, E., Wallace, E., Ritter, E., Mays, E., Wang, F., Such, F. P., Raso, F., Leoni, F., Tsimpourlas, F., Song, F., von Lohmann, F., Sulit, F., Salmon, G., Parascandolo, G., Chabot, G., Zhao, G., Brockman, G., Leclerc, G., Salman, H., Bao, H., Sheng, H., Andrin, H., Bagherinezhad, H., Ren, H., Lightman, H., Chung, H. W., Kivlichan, I., O'Connell, I., Osband, I., Gilaberte, I. C., Akkaya, I., Kostrikov, I., Sutskever, I., Kofman, I., Pachocki, J., Lennon, J., Wei, J., Harb, J., Twore, J., Feng, J., Yu, J., Weng, J., Tang, J., Yu, J., Candela, J. Q., Palermo, J., Parish, J., Heidecke, J., Hallman, J., Rizzo, J., Gordon, J., Uesato, J., Ward, J., Huizinga, J., Wang, J., Chen, K., Xiao, K., Singhal, K., Nguyen, K., Cobbe, K., Shi, K., Wood, K., Rimbach, K., Gu-Lemberg, K., Liu, K., Lu, K., Stone, K., Yu, K., Ahmad, L., Yang, L., Liu, L., Maksin, L., Ho, L., Fedus, L., Weng, L., Li, L., McCallum, L., Held, L., Kuhn, L., Kondraciuk, L., Kaiser, L., Metz, L., Boyd, M., Trebacz, M., Joglekar, M., Chen, M., Tintor, M., Meyer, M., Jones, M., Kaufer, M., Schwarzer, M., Shah, M., Yatbaz, M., Guan, M. Y., Xu, M., Yan, M., Glaese, M., Chen, M., Lampe, M., Malek, M., Wang, M., Fradin, M., McClay, M., Pavlov, M., Wang, M., Wang, M., Murati, M., Bavarian, M., Rohaninejad, M., McAleese, N., Chowdhury, N., Chowdhury, N., Ryder, N., Tezak, N., Brown, N., Nachum, O., Boiko, O., Murk, O., Watkins, O., Chao, P., Ashbourne, P., Izmailov, P., Zhokhov, P., Dias, R., Arora, R., Lin, R., Lopes, R. G., Gaon, R., Miyara, R., Leike, R., Hwang, R., Garg, R., Brown, R., James, R., Shu, R., Cheu, R., Greene, R., Jain, S., Altman, S., Toizer, S., Toyer, S., Miserendino, S., Agarwal, S., Hernandez, S., Baker, S., McKinney, S., Yan, S., Zhao, S., Hu, S., Santurkar, S., Chaudhuri, S. R., Zhang, S., Fu, S., Papay, S., Lin, S., Balaji, S., Sanjeev, S., Sidor, S., Broda, T., Clark, A., Wang, T., Gordon, T., Sanders, T., Patwardhan, T., Sottiaux, T., Degry, T., Dimson, T., Zheng, T., Garipov, T., Stasi, T., Bansal, T., Creech, T., Peterson, T., Eloundou, T., Qi, V., Kosaraju, V., Monaco, V., Pong, V., Fomenko, V., Zheng, W., Zhou, W., McCabe, W., Zaremba, W., Dubois, Y., Lu, Y., Chen, Y., Cha, Y., Bai, Y., He, Y., Zhang, Y., Wang, Y., Shao, Z., and Li, Z. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3406703. URL https://doi.org/10.1145/3394486.3406703.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Zhang, M., Li, Y., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: language agents with verbal

reinforcement learning. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023.

Team, K., Du, A., Gao, B., Xing, B., Jiang, C., Chen, C., Li, C., Xiao, C., Du, C., Liao, C., Tang, C., Wang, C., Zhang, D., Yuan, E., Lu, E., Tang, F., Sung, F., Wei, G., Lai, G., Guo, H., Zhu, H., Ding, H., Hu, H., Yang, H., Zhang, H., Yao, H., Zhao, H., Lu, H., Li, H., Yu, H., Gao, H., Zheng, H., Yuan, H., Chen, J., Guo, J., Su, J., Wang, J., Zhao, J., Zhang, J., Liu, J., Yan, J., Wu, J., Shi, L., Ye, L., Yu, L., Dong, M., Zhang, N., Ma, N., Pan, Q., Gong, Q., Liu, S., Ma, S., Wei, S., Cao, S., Huang, S., Jiang, T., Gao, W., Xiong, W., He, W., Huang, W., Xu, W., Wu, W., He, W., Wei, X., Jia, X., Wu, X., Xu, X., Zu, X., Zhou, X., Pan, X., Charles, Y., Li, Y., Hu, Y., Liu, Y., Chen, Y., Wang, Y., Liu, Y., Qin, Y., Liu, Y., Yang, Y., Bao, Y., Du, Y., Wu, Y., Wang, Y., Zhou, Z., Wang, Z., Li, Z., Zhu, Z., Zhang, Z., Wang, Z., Yang, Z., Huang, Z., Huang, Z., Xu, Z., Yang, Z., and Lin, Z. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.

von Werra, L., Belkada, Y., Tunstall, L., Beeching, E., Thrush, T., Lambert, N., Huang, S., Rasul, K., and Gallouédec, Q. TRL: Transformer Reinforcement Learning. https://github.com/huggingface/trl, 2020.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Xia, C. S., Deng, Y., Dunn, S., and Zhang, L. Demystifying llm-based software engineering agents. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754.

Xu, Y. E., Savani, Y., Fang, F., and Kolter, Z. Not all rollouts are useful: Down-sampling rollouts in llm reinforcement learning. *arXiv preprint arXiv:2504.13818*, 2025a.

Xu, Z., Liu, Y., Yin, Y., Zhou, M., and Poovendran, R. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. *arXiv preprint arXiv:2503.02951*, 2025b.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K. R., and Press, O. 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.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Yu, Q., Zhang, Z., Zhu, R., Yuan, Y., Zuo, X., Yue, Y., Dai, W., Fan, T., Liu, G., Liu, L., Liu, X., Lin, H., Lin, Z., Ma, B., Sheng, G., Tong, Y., Zhang, C., Zhang, M., Zhang, W., Zhu, H., Zhu, J., Chen, J., Chen, J., Wang, C., Yu, H., Song, Y., Wei, X., Zhou, H., Liu, J., Ma, W.-Y., Zhang, Y.-Q., Yan, L., Qiao, M., Wu, Y., and Wang, M. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Yuan, Y., Yue, Y., Zhu, R., Fan, T., and Yan, L. What's behind ppo's collapse in long-cot? value optimization holds the secret, 2025. URL https://arxiv.org/abs/2503.01491.

Yue, Y., Yuan, Y., Yu, Q., Zuo, X., Zhu, R., Xu, W., Chen, J., Wang, C., Fan, T., Du, Z., Wei, X., Yu, X., Liu, G., Liu, J., Liu, L., Lin, H., Lin, Z., Ma, B., Zhang, C., Zhang, M., Zhang, W., Zhu, H., Zhang, R., Liu, X., Wang, M., Wu, Y., and Yan, L. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks, 2025. URL https://arxiv.org/abs/2504.05118.

Zhang, S., Dong, Y., Zhang, J., Kautz, J., Catanzaro, B., Tao, A., Wu, Q., Yu, Z., and Liu, G. Nemotron-research-tool-n1: Tool-using language models with reinforced reasoning. *arXiv preprint arXiv:2505.00024*, 2025.

Zheng, C., Liu, S., Li, M., Chen, X.-H., Yu, B., Gao, C., Dang, K., Liu, Y., Men, R., Yang, A., Zhou, J., and Lin, J. Group sequence policy optimization, 2025. URL https://arxiv.org/abs/2507.18071.

Zhong, L., Wang, Z., and Shang, J. 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.

Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., and Wang, Y.-X. Language agent tree search unifies reasoning, acting, and planning in language models. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=njwv9BsGHF.

Zhuang*, R., Vu*, T., Dimakis, A., and Sathiamoorthy, M. Improving multi-turn tool use with reinforcement learning, 2025. Accessed: 2025-04-17.

Zhuo, T. Y., Chien, V. M., Chim, J., Hu, H., Yu, W., Widyasari, R., Yusuf, I. N. B., Zhan, H., He, J., Paul, I., Brunner, S., GONG, C., Hoang, J., Zebaze, A. R., Hong, X., Li, W.-D., Kaddour, J., Xu, M., Zhang, Z., Yadav, P., Jain, N., Gu, A., Cheng, Z., Liu, J., Liu, Q., Wang, Z., Lo, D., Hui, B., Muennighoff, N., Fried, D., Du, X., de Vries, H., and Werra, L. V. 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.