Reinforcement Learning For Code Synthesis

Zihan Wang , Jiajun Xu, and Lei Wu

2024210885, 2024210900, 2024210897

Department of Computer Science and Techonology, Tsinghua University

{w-zh24, xu-jj24, wulei24}@mails.tsinghua.edu.cn

Abstract

Large Language Models (LLMs) have achieved significant success in programming tasks, particularly excelling on interview-oriented platforms like LeetCode. However, we observe that these models still underperform on more complex problems in Olympiad level competitions. This performance gap primarily stems from the deep mathematical reasoning, complex algorithmic thinking, and diverse solution strategies required for Olympiad programming problems. To address this issue, we propose a novel approach: Reinforcement Learning with Online Judge Feedback (RLOJF). This method simulates the iterative process in real programming environments, allowing the model to dynamically adjust its output based on scores and error messages provided by Online Judge (OJ) systems. RLOJF aims to improve the correctness and efficiency of code generated by the model, develop its ability to iteratively refine code using automated feedback, and enhance its reasoning and problem-solving capabilities in complex programming tasks. Our research contributions include: proposing a new reinforcement learning framework for complex programming tasks, designing a training methodology utilizing OJ feedback, conducting extensive experiments on a large number of complex programming problems to validate the method's effectiveness.

1 Background

Code synthesis, or the automatic generation of code, has gained significant attention in AI research due to its potential to streamline software development. Traditional methods and even recent machine learning models like transformers often struggle with complex or unfamiliar tasks, relying on predefined rules or supervised learning from fixed examples. Reinforcement Learning (RL) offers a promising alternative, allowing systems to learn by interacting with an environment and receiving feedback, potentially becoming more flexible and creative in generating code.

Olympiad-level programming tasks present an even greater challenge, requiring consideration of complex systems design, advanced mathematical theories, sophisticated algorithms, and optimization for both time and space complexity. These tasks are significantly more difficult than typical interview coding problems, and even top-tier competitive programmers rarely solve them correctly on the first attempt. The problem-solving process for such tasks can be abstracted into agent-like behaviors: compiling code, trying custom test cases, and submitting to online judge for black-box feedback.

We propose the Reinforcement Learning with Online Judge Feedback (RLOJF) framework, which leverages the advantages of online reinforcement learning: real-time adaptation, efficient exploration, robustness to problem diversity, and mimicking human learning processes. By combining the pattern recognition capabilities of LLMs with the adaptive learning of RL and the rapid feedback of advanced Online Judge systems, RLOJF represents a significant step forward in AI-assisted programming for

Preprint. Under review.

high-level competitive coding scenarios, pushing the boundaries of what's possible in automated code synthesis for complex programming tasks.

2 Definition

Reinforcement Learning (RL) is a machine learning technique where an agent learns by interacting with an environment and receiving rewards or penalties based on its actions. The goal is to maximize the total reward over time by making the best decisions.

In the context of code synthesis, RL offers a promising approach to automate and improve the process of generating computer programs. The application of RL to code generation involves casting the problem into the RL framework: the environment is defined by the rules and structure of the target programming language, the actions correspond to the selection of code elements or transformations, and the rewards are designed to reflect the quality of the generated code. This quality can be measured in various ways, such as correctness, efficiency, and readability. By framing code synthesis as an RL problem, we can leverage the power of modern RL algorithms to navigate the vast and complex search space of possible programs.

3 Related works

3.1 Proximal Policy Optimization

Proximal Policy Optimization(PPO) [1] is a popular reinforcement learning algorithm designed to improve the stability and efficiency of policy updates, which uses a clipped objective function to restrict how much the policy can change during updates, ensuring that updates remain "proximal" to the previous policy.

3.2 Reinforcement Learning with Execution Feedback

Reinforcement Learning with Execution Feedback(RLEF)[2] is a recent reinforcement learning method proposed to teach models to leverage execution feedback for code improvement, which achieves this by optimizing reward signals within an iterative environment.

Experimental results on the CodeContests benchmark demonstrate that the RLEF training method achieves state-of-the-art results for both small (8B parameters) and large (70B parameters) models, while reducing the number of samples required.

4 Method

RLOJF framework consists of a high-performance OJ system that compiles, executes, and evaluates submitted code against comprehensive test cases, providing real-time feedback including runtime, memory metrics, and test results. We use open-source OJ systems to build our environment[1]. Besides the commonly used code evaluation dataset CodeContest[3], we also introduce an additional Chinese OI (Olympiad in Informatics) dataset based on Chinese high school OI competitions. We plan to test the effectiveness of our approach on LLAMA 3.1[4] and GLM-4-9B[5] models. Specifically, we aim to enhance Chinese programming capabilities on the GLM-4-9B model.

For the RL part, we define the state space $\mathcal{S}$ as a tuple (c, r, f), where c represents the current code, r denotes compilation and test results, and f encompasses OJ feedback such as scores and memory usage. The action space $\mathcal{A}$ is designed to include code generation, modification, compilation, and submissions to the OJ. To guide the learning process, the RLOJF agent optimizes a reward function $R(s, a, s')$ that promotes correctness, efficiency, and continuous improvement in the generated code. This reward function is formulated as:

$$R(s, a, s') = w_c C(s') + w_p P(s') + w_i I(s, s') - w_e E(s') - w_t T(s') - w_m M(s')$$

where C, P, I, E, T, and M respectively represent correctness, performance, improvement, error, time, and memory usage, each weighted by corresponding factors w to balance these competing objectives.

[1] https://hydro.ac/, https://loj.ac/

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[2] Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. Rlef: Grounding code llms in execution feedback with reinforcement learning. *arXiv preprint arXiv:2410.02089*, 2024.

[3] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[4] Llama Team AI @ Meta. The Llama 3 Herd of Models. Technical report, 2024.

[5] Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

[6] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

[7] Zhaofeng Liu, Jing Su, Jia Cai, Jingzhi Yang, and Chenfan Wu. Instruct-code-llama: Improving capabilities of language model in competition level code generation by online judge feedback. In *International Conference on Intelligent Computing*, pages 127–137. Springer, 2024.

[8] Kunhao Zheng, Juliette Decugis, Jonas Gehring, Taco Cohen, Benjamin Negrevergne, and Gabriel Synnaeve. What makes large language models reason in (multi-turn) code generation? *arXiv preprint arXiv:2410.08105*, 2024.

[9] Zhi-Cun Lyu, Xin-Ye Li, Zheng Xie, and Ming Li. Top pass: Improve code generation by pass@ k-maximized code ranking. *arXiv preprint arXiv:2408.05715*, 2024.

[10] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.