International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.com | editor@ijrai.com | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 8, Issue 6, September–October 2025||

DOI:10.15662/IJRAI.2025.0806003

Autonomous Code Generation with Reinforcement-Learning-Based LLMs

Amit Rajiv Malhotra

City Engineering College, Bengaluru, Karnataka, India

ABSTRACT: The integration of reinforcement learning (RL) with large language models (LLMs) has significantly advanced autonomous code generation. Traditional supervised learning approaches often overlook the complexities inherent in code synthesis, such as functional correctness and adherence to coding standards. Reinforcement learning addresses these challenges by enabling models to learn from feedback, optimizing code generation processes. This paper explores the synergy between RL and LLMs in code generation, highlighting key methodologies, advancements, and applications. We examine frameworks like CodeRL, which employs actor-critic architectures to refine code outputs through functional correctness feedback. Additionally, we discuss the role of execution-based feedback, as seen in PPOCoder, which utilizes Proximal Policy Optimization to enhance code generation. The paper also delves into the concept of reinforcement learning from human feedback (RLHF), focusing on its application in aligning LLM outputs with human preferences. Furthermore, we address the challenges associated with these approaches, including the need for diverse training data and the complexities of reward signal design. Through a comprehensive review, this paper provides insights into the current landscape of RL-enhanced LLMs for code generation and outlines directions for future research.arXiv+2GitHub+2arXiv+1Wikipedia+1

KEYWORDS: Autonomous Code Generation, Reinforcement Learning, Large Language Models, Code Synthesis, Functional Correctness, Execution-Based Feedback, Human Feedback Alignment, Code Optimization.

I. INTRODUCTION

The evolution of large language models (LLMs) has revolutionized various domains, including natural language processing and code generation. However, generating code that is not only syntactically correct but also functionally accurate and efficient remains a significant challenge. Traditional supervised learning methods, which rely on vast datasets of code examples, often fall short in capturing the nuances of code semantics and the intricacies of programming logic. Reinforcement learning (RL) offers a promising solution by enabling models to learn optimal coding strategies through feedback mechanisms.

In the context of code generation, RL can be employed to refine LLM outputs by providing a structured approach to learning from execution results. Frameworks like CodeRL utilize actor-critic architectures, where the model's code generation capabilities are enhanced by feedback on the functional correctness of the generated code. Similarly, execution-based feedback systems, such as PPOCoder, leverage reinforcement learning techniques to optimize code generation processes.arXivarXiv+3arXiv+3GitHub+3arXiv+1

Moreover, the integration of human feedback into the training process, known as reinforcement learning from human feedback (RLHF), has been shown to align LLM outputs more closely with human preferences and expectations. This approach not only improves the quality of generated code but also enhances its readability and maintainability.arXiv+1 Despite these advancements, several challenges persist, including the design of effective reward signals, the need for diverse and representative training data, and the complexities associated with aligning model outputs with human intent. Addressing these issues is crucial for the development of autonomous code generation systems that can meet the demands of modern software development.

II. LITERATURE REVIEW

The intersection of reinforcement learning and large language models for code generation has been a focal point of recent research. Le et al. (2022) introduced CodeRL, a framework that employs a pretrained LLM as an actor and a critic network to assess the functional correctness of generated code. This approach demonstrated significant

| ISSN: 2455-1864 | www.ijrai.com | editor@ijrai.com | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 8, Issue 6, September–October 2025||

DOI:10.15662/IJRAI.2025.0806003

improvements in code generation tasks, particularly in benchmarks like APPS and MBPP. Wikipedia+4arXiv+4arXiv+4

Building upon this, Shojaee et al. (2023) proposed PPOCoder, which integrates Proximal Policy Optimization with execution-based feedback to enhance code generation. By utilizing non-differentiable feedback from code execution, PPOCoder effectively addresses challenges related to code compilation and functional correctness. arXiv+2GitHub+2 Further advancements were made by Jain et al. (2023), who introduced RLCF, a method that refines LLMs using reinforcement learning feedback derived from compiler outputs and comparative analysis with reference code. This approach improved the likelihood of generated code compiling and producing correct outputs, even with smaller model sizes. arXiv+1

Incorporating human feedback, Bai et al. (2022) and Lee et al. (2023) explored reinforcement learning from human feedback (RLHF), where models are fine-tuned based on human preferences. This methodology has been instrumental in enhancing the alignment of LLM outputs with human expectations, leading to more reliable and user-aligned code generation.

Despite these advancements, challenges remain in designing effective reward functions, ensuring the diversity of training data, and aligning model outputs with human intent. Addressing these issues is essential for the continued progress in autonomous code generation.

III. RESEARCH METHODOLOGY

This study employs a systematic review methodology to analyze existing literature on the integration of reinforcement learning with large language models for code generation. The review process involves the following steps:

Literature Search: Comprehensive searches were conducted in academic databases such as arXiv, IEEE Xplore, and Google Scholar using keywords like "reinforcement learning," "large language models," "code generation," and "autonomous programming."

Selection Criteria: Studies included in the review were published before 2022 and focused on the application of reinforcement learning in code generation tasks. Both theoretical and empirical studies were considered to provide a holistic view of the field.

Data Extraction: Key information was extracted from selected studies, including methodologies, frameworks, benchmarks used, and outcomes achieved.

Analysis and Synthesis: The extracted data were analyzed to identify common themes, methodologies, and challenges. Comparative analysis was conducted to evaluate the effectiveness of different approaches.

Reporting: The findings were synthesized into a comprehensive report, highlighting the state of research, advancements made, and areas requiring further investigation.

Advantages

1. **Enhanced Functional Correctness**: Reinforcement learning (RL) enables models to learn from feedback, improving the functional accuracy of generated code. For instance, CodeRL employs a critic network to assess the correctness of generated programs, leading to higher performance on benchmarks like APPS and MBPP. arXiv

2. **Adaptability to Complex Tasks**: RL-based models can handle complex coding tasks by learning from execution feedback. PPOCoder, for example, utilizes Proximal Policy Optimization to refine code generation, achieving significant improvements in compilation success rates and functional correctness across different programming languages. arXiv

3. **Alignment with Human Preferences**: Reinforcement learning from human feedback (RLHF) allows models to align their outputs with human expectations, enhancing the quality and readability of generated code. This approach has been shown to improve the likelihood of generated code compiling and producing correct outputs.

International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.com | editor@ijrai.com | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 8, Issue 6, September–October 2025||

DOI:10.15662/IJRAI.2025.0806003

Disadvantages

1. **Sample Inefficiency**: RL algorithms often require a large number of interactions with the environment to learn effective policies, leading to high computational costs and time-intensive training. Wikipedia

2. **Stability and Convergence Issues**: Training RL models can be unstable and prone to divergence, making it difficult to achieve consistent results. This instability is further enhanced in the case of continuous or high-dimensional action spaces. Wikipedia

3. **Designing Effective Reward Functions**: Creating appropriate reward functions is critical in RL because poorly designed reward functions can lead to unintended behaviors and suboptimal performance. Wikipedia

IV. RESULTS AND DISCUSSION

Studies have demonstrated the effectiveness of RL-based approaches in autonomous code generation. CodeRL achieved state-of-the-art results on the APPS benchmark, outperforming traditional supervised learning methods. Similarly, PPOCoder showed significant improvements in compilation success rates and functional correctness across different programming languages. These advancements highlight the potential of RL in addressing the challenges of code generation, such as ensuring functional correctness and adapting to complex tasks. arXiv

V. CONCLUSION

Reinforcement learning enhances the capabilities of large language models in autonomous code generation by improving functional correctness, adaptability, and alignment with human preferences. While challenges such as sample inefficiency, stability issues, and reward function design remain, ongoing research and advancements in RL techniques continue to address these limitations, paving the way for more effective and reliable code generation systems.Wikipedia

VI. FUTURE WORK

1. **Improving Sample Efficiency**: Developing methods to reduce the number of interactions required for RL models to learn effective policies can make training more practical and less resource-intensive.Wikipedia
2. **Enhancing Stability and Convergence**: Implementing techniques to stabilize training and ensure convergence can lead to more reliable and consistent performance in RL-based code generation models.
3. **Designing Robust Reward Functions**: Creating reward functions that accurately reflect the desired outcomes can help in guiding RL models towards generating high-quality code.

REFERENCES

1. Le, H., Wang, Y., Gotmare, A. D., Savarese, S., & Hoi, S. C. H. (2022). CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. *arXiv*.
2. Shojaee, P., Jain, A., Tipirneni, S., & Reddy, C. K. (2023). Execution-based Code Generation using Deep Reinforcement Learning. *arXiv*.
3. Jain, A., Adiole, C., Chaudhuri, S., Reps, T., & Jermaine, C. (2023). Coarse-Tuning Models of Code with Reinforcement Learning Feedback. *arXiv*.
4. Wikipedia contributors. (2023). Reinforcement learning. *Wikipedia*.
5. Wikipedia contributors. (2023). Reinforcement learning from human feedback. *Wikipedia*.
6. Kashyap, A. (2024). Harnessing Advanced Reinforcement Learning and Transformers: Shaping the Future of Autonomous Generation. *Medium*.
7. Rasheed, Z., Sami, M. A., Kemell, K.-K., Waseem, M., Saari, M., Systä, K., & Abrahamsson, P. (2024). CodePori: Large-Scale System for Autonomous Software Development Using Multi-Agent Technology. *arXiv*.
8. Wikipedia contributors. (2023). Reinforcement learning. *Wikipedia*.
9. Wikipedia contributors. (2023). Reinforcement learning from human feedback. *Wikipedia*.