Advancing Multi-Agent Reinforcement Learning for Collaborative Coding Agents

*Kanav Gupta
SCAI
Arizona State University
Tempe, Arizona, USA
kgupta72@asu.edu

*Dhruv Bansal
SCAI
Arizona State University
Tempe, Arizona, USA
dbansa11@asu.edu

*Sameer Kamble
SCAI
Arizona State University
Tempe, Arizona, USA
srkamble@asu.edu

* Equal Contribution

Abstract

Complex programming tasks challenge single-agent LLM systems since they require deep understanding, planning, and iterative debugging. Most existing multi-agent frameworks depend on static prompting or manually designed pipelines, struggling with dynamic communication, credit assignment, and scalability.

We develop a multi-agent framework driven by reinforcement learning for cooperative coding agents. A Centralized Training Decentralized Execution (CTDE) approach enables multiple LLMs to collaborate in efficiently planning and executing coding tasks.

Our aim is to improve the traditional LLM's and Agent implementation using the complexity of Reinforcement Learning in a controlled environment.

1 Introduction

Large Language Models (LLMs) demonstrate strong capabilities in natural language reasoning and code synthesis, but single-agent systems often struggle with complex programming tasks requiring decomposition, multi-step planning, iterative debugging, or feedback integration. Multi-agent LLM systems have recently emerged as a promising direction, enabling role specialization, communication, and collaborative decision-making [1, 2, 3].

However, most systems, such as MapCoder [3], ChatDev [2], and AutoGen [1], depend heavily on manually engineered pipelines rather than learning-based coordination. Reinforcement Learning (RL), especially in multi-agent settings, has not been widely applied to collaborative coding, with only limited progress from frameworks like Co-Learning [4], MAGRPO [5], and TF-GRPO [6].

We fill this gap by designing a reinforcement-learning-based multi-agent framework that enables LLM agents to coordinate adaptively using CTDE, collaborative decision-making, and shared rewards driven by test-case outcomes. We enhance the MARL architecture by employing two agents (helper and main) and using Multi-Agent Group Relative Policy Optimization (MAGRPO) that trains multiple LLMs jointly through centralized group-relative advantages while maintaining decentralized execution efficiency.

Modeling LLM collaboration as a cooperative MARL problem and training with MAGRPO yields adaptive, sample-efficient systems capable of solving challenging programming tasks through learned coordination strategies rather than hand-engineered pipelines.

2 Related Work

2.1 Multi-Agent Collaboration

Multi-agent LLM frameworks demonstrate that structured roles and communications can improve code generation and debugging. AutoGen [1] enables multi-agent conversation but relies on prompt engineering rather than adaptive learning. MapCoder [3] uses specialized agents for retrieval, planning, coding, and debugging, achieving strong benchmark results but following a fixed pipeline. ChatDev [2] models a full software development workflow using designer, coder, tester, and documenter agents, but lacks dynamic coordination. Co-Learning [4] introduces test-driven reinforcement to choose agents based on code correctness, though it depends on handcrafted rewards and a small dataset.

Communication and role specialization are effective, but none of these frameworks support learning-based scheduling, credit assignment, or scalable adaptive coordination.

2.2 Reinforcement Learning in Multi-Agent LLM Systems

Few studies apply reinforcement learning to collaborative LLM coding. Co-Learning [4] uses Environmental RL to reward correct outputs but does not train agents jointly. Training-Free GRPO [6] offers lightweight relative policy updates without pretraining, reducing computation but remaining limited to synthetic benchmarks.

These approaches show early success but do not yet scale to heterogeneous coding agents, motivating the need for a CTDE-driven multi-agent framework.

3 Data

Our project uses the HumanEval dataset to support code generation evaluation, multi-agent coordination, and reinforcement learning for debugging.

3.1 HumanEval

Source: OpenAI HumanEval Benchmark
Purpose: Evaluating Functional correctness of generated Python code using unit tests
Size: 164 programming tasks and test cases

Why We Use It: HumanEval serves as the standard benchmark for assessing LLM coding performance. It allows direct comparison with established multi-agent systems such as MapCoder (Islam et al., 2024) and MAGRPO (Liu et al., 2025).

Dataset Split: HumanEval is typically evaluated as a fixed test set. We follow a standard practice:

- 75.76% train (25/33 ratio)
- 24.24% test (8/33 ratio)

4 Methods

Our approach combines the multi-agent architecture from Co-Learning with the Multi-Agent Group Relative Policy Optimization (MAGRPO) algorithm for end-to-end reinforcement learning. We conduct a systematic evaluation progressing from baseline single-agent systems to fully coordinated multi-agent systems with reinforcement learning.

Our system consists of two specialized agents (main and helper) that work together to solve programming problems. Each agent operates with partial knowledge of the complete system state, observing different aspects of the problem-solving process based on its role.

4.1 Multi-Agent Architecture

Our system instantiates two specialized LLM agents, each with distinct roles in the collaborative coding workflow:

Helper Agent (Agent 0): Generates utility/helper functions that decompose the problem into manageable sub-components. The helper agent receives the problem description and entry point, then produces auxiliary functions that the main agent can leverage.

Main Agent (Agent 1): Receives both the original problem prompt and the helper code generated by Agent 0. It then completes the target function, potentially utilizing the helper functions to construct the final solution.

4.2 MAGRPO Algorithm

We adapt Group Relative Policy Optimization (GRPO) to multi-agent settings as described in [5]. At each training episode:

1. Sample a problem and initialize state
2. For each turn $t \in [0, H-1]$:
   a. Generate G diverse solutions per agent
   b. Execute actions and compute rewards
   c. Store trajectory data

3. Compute group-relative advantages: $A_t^{(g)} = R_t^{(g)} - \text{mean}(R_t)$
4. Update policies using policy gradient with loss: $L = -A \cdot \log \pi\theta(a|h)$

We use Qwen2.5-Coder-3B as our base model. Our implementation differs from [5] in several ways: they use Qwen2.5-Coder-7B (larger model), a tree-like structure for rollouts, discounted returns with configurable discount ($\gamma = 0.9$), and multiple optimizer steps per tree node.

5 Experiments

Hardware: 1x NVIDIA A100 (40GB or 80GB), **2× AMD EPYC 7413 processors** (24 cores each), 503 GiB RAM total, Rocky Linux 8.10 (Green Obsidian)
Base Model: Qwen2.5-Coder-3B (bfloat16)
Dataset: HumanEval

5.1 Evaluation Protocol

1. **Baseline Single Model:** The base Qwen2.5-Coder-3B model without any fine-tuning. For each problem, we generate k independent solutions using temperature sampling (temperature=0.8) via the vLLM inference server. Each solution is generated from the problem prompt alone, with code extracted from the model's response. This establishes our baseline Pass@k performance for a single untrained agent.

2. **Baseline Multi-Agent:** Two instances of the same Qwen2.5-Coder-3B model working in sequence without RL training. Agent 0 (helper) receives a specialized prompt asking it to generate a helper function for the problem. Agent 1 (main) then receives both the original problem and the helper's generated code, completing the target function. This tests whether role decomposition through prompting alone improves performance over the single-agent baseline.

3. **GRPO Single-Agent:** A single Qwen2.5-Coder-3B model trained using Group Relative Policy Optimization. The model generates *G* candidate solutions per problem, receives rewards based on code correctness (+0.5 for runnable code, +0.1 per passed test up to 5 tests), and updates using group-relative advantages. Training runs for multiple epochs over the training split, with the final model evaluated on the test set.

4. **Multi-Agent GRPO:** Our full system with two specialized agents (helper and main) trained jointly using MAGRPO. Both agents share the same reward signal based on the combined code's correctness. The helper agent learns to generate useful auxiliary functions while the main agent learns to leverage them effectively. Rewards are computed on the combined output: +0.2 for syntactically valid code, +0.3 for runnable code, and +0.1 per passed test. Agents are updated using centralized group-relative advantages while maintaining decentralized execution.

5.2 Metrics

Pass@k: Probability that at least 1 of *k* solutions passes all tests:

$$\text{pass@}k = \mathbb{E}_{\text{problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]$$

where *n* is the total number of samples, *c* is the number of correct samples, and *k* is the number of samples considered.

6 Results

Method	Pass@1	Pass@3	Pass@5	Pass@10
Single Model	29.25%	53.02%	63.11%	72.50%
Multi-Agent	24.00%	49.50%	61.94%	75.00%
GRPO	14.50%	30.96%	40.96%	55.00%
MAGRPO	**44.25%**	**65.23%**	**72.29%**	**80.00%**

Table 1: Results on 40/164 HumanEval problems with 10 samples per problem.

6.1 Discussion

Our results reveal several key insights:

1. **Multi-Agent MAGRPO achieves the best performance** across all Pass@k metrics, with a 51% relative improvement in Pass@1 over the single-agent baseline (44.25% vs 29.25%).

2. **Baseline multi-agent underperforms** the single-agent baseline at lower k values (24% vs 29.25% at Pass@1), suggesting that naive role decomposition through prompting alone can introduce coordination overhead.

3. **Single-agent GRPO shows degraded performance,** likely due to small model size and limited training data, which may cause mode collapse. This could also be attributed towards small group size (G) of only 4.

4. **MAGRPO's joint training is important.** The improvement from baseline multi-agent to MAGRPO shows that learned coordination significantly outperforms static prompting-based collaboration.

7 Conclusion

We presented a multi-agent reinforcement learning framework for collaborative code generation using MAGRPO. Our results demonstrate that jointly training specialized agents (helper and main) with shared rewards substantially outperforms both single-agent baselines and prompt-based multi-agent systems.

The key finding is that learned coordination through MAGRPO enables effective collaboration that static role assignment cannot achieve. While baseline multi-agent systems may struggle with coordination overhead, RL training allows agents to develop complementary strategies that leverage each other's outputs effectively.

Future work includes scaling to larger models, exploring more complex agent hierarchies, and extending evaluation to additional benchmarks.

References

1. Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Ap-plications via Multi-Agent Conversation," *arXiv preprint arXiv:2308.08155*, 2023.
2. C. Qian et al., "ChatDev: Communicative Agents for Software Development," *arXiv preprint arXiv:2307.07924*, 2023.
3. M. A. Islam et al., "MapCoder: Multi-Agent Code Genera-tion for Competitive Problem Solving," in *Proc. 62nd Annual Meeting of the ACL*, 2024.
4. J. Yu et al., "Co-Learning: Code Learning for Multi-Agent Reinforcement Collaborative Framework," *arXiv preprint arXiv:2409.00985*,

2025.

5. S. Liu et al., “LLM Collaboration with Multi-Agent Reinforce-ment Learning,” *arXiv preprint arXiv:2508.04652*, 2025.
6. Y. Cai et al., “Training-Free Group Relative Policy Optimiza-tion,” *arXiv preprint arXiv:2510.08191*, 2025.
7. M. Chen et al., “Evaluating Large Language Models Trained on Code,” *arXiv preprint arXiv:2107.03374*, 2021.
8. S. Lu et al., “CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation,” in *NeurIPS Datasets and Benchmarks Track*, 2021.