Tree-Augmented RL for Autonomous Repair of AI-Generated Code

Ozerova Daria

20.04.2025

Project proposal

LLMs in Code Refinement Statements:

- **Self-debug ability** is crucial for LLMs to refine their generated code based on execution feedback, which is important for solving complex problems.
- Iterative code refinement does not involve a trade-off between **exploration and exploitation**.

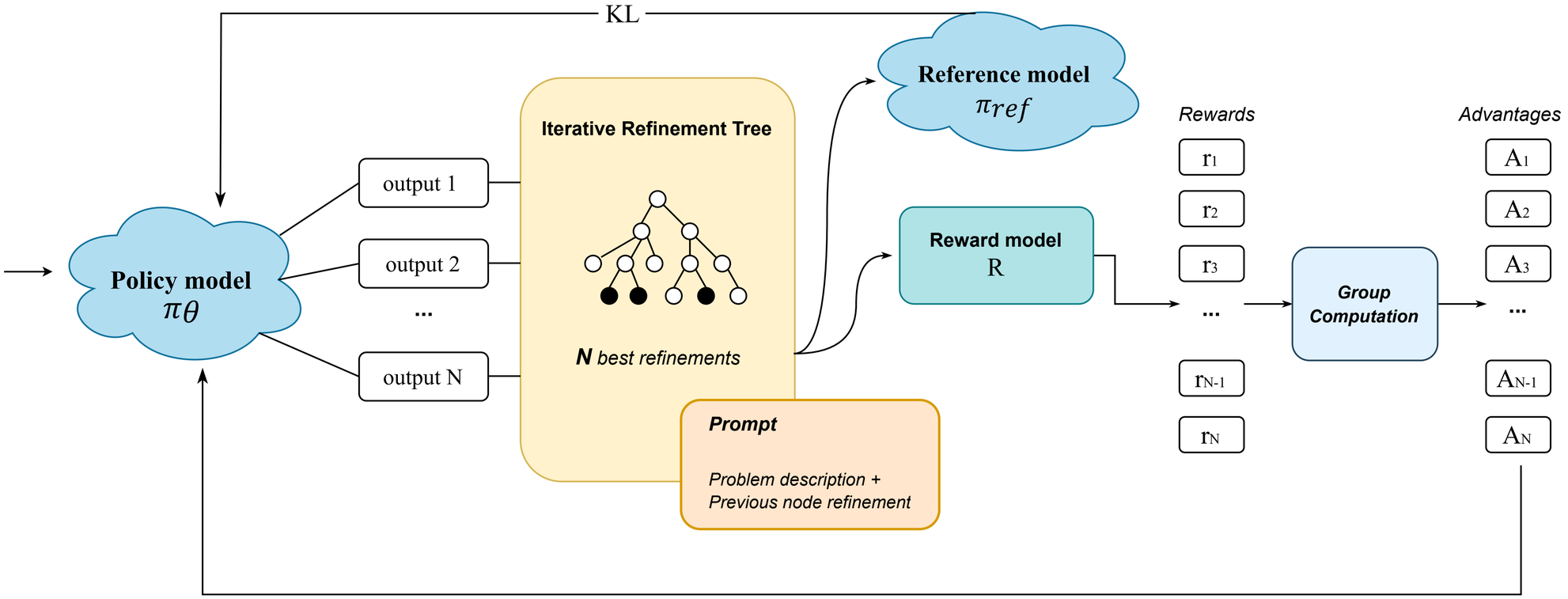
Key Contributions:

- **Syntax-Aware Reinforcement Learning Pipeline**
  We present a novel approach that integrates reinforcement learning with structured code analysis, featuring a **custom syntax-tree reward function** that improves output correctness while maintaining training stability.
- **Benchmark Evaluation**
  Experimental results on MBPP, APPS and HumanEval demonstrate:
  - **5% improvement** in code repair accuracy over baseline methods
  - Optimization algorithmic complexity while maintaining correct behavior

Overview of existing solutions

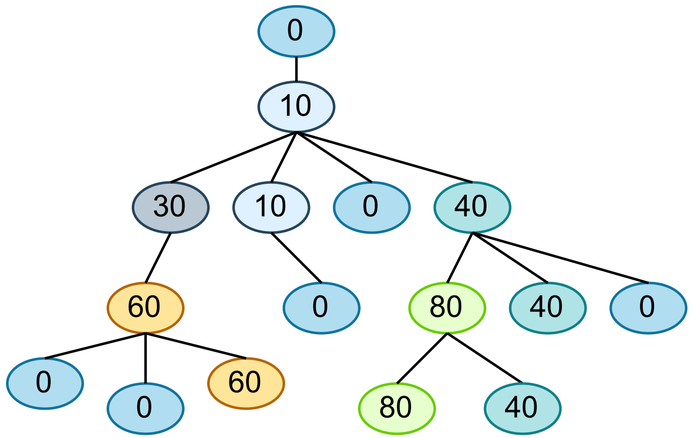
Paper	Solution method	Models	Evaluation details	Benchmarks	Metrics
Tang H. et al. Code repair with llms gives an exploration-exploitation tradeoff. Advances in Neural Information Processing Systems. – 2024. – Т. 37. – С. 117954-117996.	Proposes *REx*, a Thompson Sampling-based algorithm for iterative code refinement, balancing exploration and exploitation	GPT-4, GPT-3.5-turbo, Claude-3.5-Sonnet, Llama-3.1-405B	Tests on loop invariant synthesis, visual reasoning (ARC), and competition programming (APPS), comparing against greedy, BFS, and fixed-width baselines	APPS (Competition, Interview, Introductory), ARC subset, and nonlinear loop invariant synthesis tasks	Passing rate, LLM call efficiency (speedup), and hyperparameter sensitivity
Zhong L., Wang Z., Shang J. Debug like a human: A large language model debugger via verifying runtime execution step-by-step arXiv:2402.16906. – 2024.	LDB uses runtime execution traces and NLP-based block-level analysis to verify and refine LLM-generated code	GPT-3.5, CodeLlama, and StarCoder	Iterative debugging with up to 10 iterations	HumanEval, MBPP, and TransCoder	Pass@k rate
Jiang N. et al. LeDex: Training LLMs to Better Self-Debug and Explain Code // Advances in Neural Information Processing Systems. – 2024. – Т. 37. – С. 35517-35543.	LeDex trains LLMs via SFT and RL with automated data collection (explanations + refinements) and execution-based filtering	Evaluated on StarCoder-15B, CodeLlama-7B/13B, with GPT-3.5/CodeLlama-34B as data sources	Iterative refinement (up to 3 rounds), tested on initial and refined code with pass@k and success rate metrics	MBPP, HumanEval, MBPP+, HumanEval+	Pass@1 (up to +15.92%), pass@10 (up to +9.30%), and refinement success rate (up to +10.15%)
Bi Z. et al. Iterative refinement of project-level code context for precise code generation with compiler feedback //arXiv preprint arXiv:2403.16792. – 2024.	COCoGen combines static analysis, compiler feedback, and iterative retrieval of project-specific context to automatically detect and fix errors in LLM-generated code	GPT-3.5-Turbo and Code Llama (13B)	Focused on class/file/project-runnable tasks in real-world repositories	CoderEval, HumanEval, MBPP, and CrossCodeEval	Pass@k rate

Proposed solution: GRPO + Iterative Refinement Tree



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

Technical details: Iterative Refinement Tree



Example or refinement tree:
The value in each node represents the percentage of passing tests for the current refinement

- **Reward function R definition:**
  - r - refinement
  - r_c - correct refinement (canonical solution from benchmark)
  - T_p - number of passed tests, T - all tests count

$$R(r) = \text{CodeBLEU}(r, r_c) + \frac{|T_p(r)|}{|T|}$$

- **Choosing next program ρ_{next} to refine:**
  - C - hyperparameter
  - N_p - number of times program ρ was refined with no reward

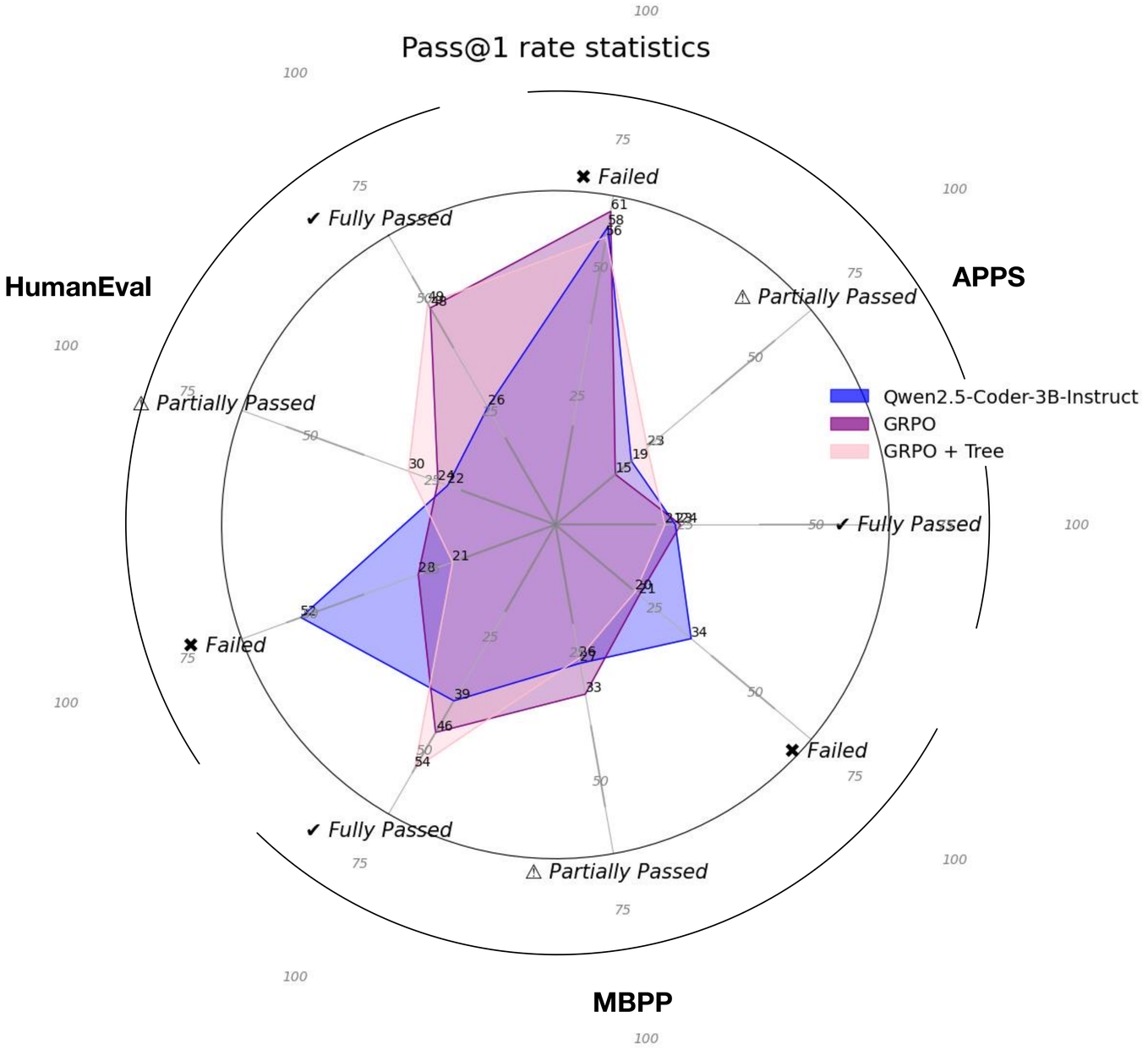
$$\rho_{\text{next}} = \arg\max_{\rho} \theta_{\rho}^{(s)}, \quad \text{where } \theta_{\rho}^{(s)} \sim \text{Beta}(\alpha_{\rho}, \beta_{\rho})$$

$$\alpha_{\rho} = 1 + C \cdot R(\rho)$$

$$\beta_{\rho} = 1 + C \cdot (1 - R(\rho)) + N_{\rho}$$

Results

- **Reference model / Policy model:**
  Qwen2.5-Coder-3B-Instruct
- **Reward model:**
  Qwen2.5-Coder-3B-Instruct fine-tuned on custom reward from previous slide
- **Datasets**: HumanEval, MBPP, APPS (available on HF)



** Each axis represents a test dataset and is scaled to 100%, corresponding to the size of the test set. The colored regions indicate the percentage of programs (relative to the test dataset size) that fall into each category.*

Results

Future directions:

- **Automated Test Generation**
  Develop a self-validating refinement pipeline that automatically generates verification tests for corrected code.
- **Error Explanation Integration**
  Enhance the model's output with natural language explanations of detected errors and proposed fixes.
- **Optimization for further integration**
  Reduce computational overhead to enable **integration into IDEs and CI/CD pipelines**.

Code is available: https://github.com/dzrlva/SMILES2025_project