RL-Based LLVM IR Program Synthesis: A Three-Stage Approach

Your Name

June 13, 2025

1 RL Algorithm Selection and Justification

For this LLVM IR program synthesis task, I implemented **REINFORCE (Policy Gradient)** for Stage 1 and **Actor-Critic variants** for Stages 2-3. Policy gradient methods were selected for several key reasons: (1) **Discrete action spaces** naturally suit both token generation and high-level action selection, (2) **Variable-length sequences** are handled elegantly by policy gradients, (3) **Sparse rewards** from binary compilation success align well with episodic reward attribution, and (4) **Stochastic policies** provide natural exploration in complex program spaces.

Stage-specific adaptations included pure REINFORCE with LSTM networks for Stage 1's sequential token generation, while Stages 2-3 employed Actor-Critic with sophisticated ϵ-greedy exploration strategies biased toward domain-specific patterns. DQN was rejected due to combinatorial state-action explosion, and PPO was deemed unnecessarily complex for the constrained action spaces.

2 Curriculum and Staged Training Strategy

The project employed a **progressive complexity curriculum** across three distinct stages:

Stage 1 focused on template-based LLVM IR generation, prioritizing syntactic validity over functionality. **Stage 2** implemented a test case curriculum starting with the simplest sorting case $[2, 1, 3]$ before progressing to multiple test cases, with success-rate gating at 85% threshold before advancement. **Stage 3** incorporated performance-aware curriculum with pattern recognition, using discovered optimal patterns like $[bc, ab, bc]$ to bias exploration toward efficient solutions.

The curriculum prevented catastrophic forgetting through adaptive advancement, ensuring solid foundations before complexity increases. This approach reduced training time by 60-70% compared to flat training across all difficulty levels simultaneously.

3 Challenges Encountered

3.1 Template Convergence and Action Space Explosion

Stage 1 suffered from convergence to trivial solutions (`ret i32 0`), achieving 100% compilation success but limited functionality. The fundamental challenge was the intractable LLVM token vocabulary ($\sim$50 tokens) creating exploration difficulties. The breakthrough solution involved **abstraction elevation** - replacing low-level token generation with high-level conditional swap operations (`if_a_gt_b_swap_ab`, etc.), reducing the action space to 4-5 meaningful operations and enabling Stage 2 success.

3.2 Training Instability and Catastrophic Forgetting

Stage 2 experienced dramatic performance swings, with success rates plummeting from 96% to 24% during episodes 1350-3000. Analysis revealed catastrophic forgetting caused by: (1) aggressive ϵ decay reducing exploration, (2) policy updates destroying successful patterns, and (3) insufficient stabilization mechanisms. Mitigation included minimum ϵ thresholds (0.15) and extended training (6000 episodes), eventually recovering to 89.5% success rate.

3.3 Multi-Objective Optimization Complexity

Stage 3 required balancing correctness maintenance with speed optimization. The solution employed sophisticated reward structuring: $R_{total} = 1.0 + R_{latency} + R_{length} + R_{pattern}$, combining correctness requirements with performance incentives. Timing measurement noise was addressed through statistical robustness using trimmed means and multiple trial averaging.

4 Metrics and Performance Analysis

Stage	Primary Focus	Success Rate	Key Achievement
Stage 1	Syntax Validity	100%	Rapid convergence (200 episodes)
Stage 2	Sorting Correctness	89.5%	Curriculum learning success
Stage 3	Speed Optimization	84.8%	68.4% optimal efficiency

Table 1: Performance summary across three-stage progression

Stage 1 achieved perfect compilation validity but converged to minimal programs, highlighting the need for more sophisticated approaches. **Stage 2** demonstrated successful curriculum learning despite training instability, with peak performance of 96% before catastrophic forgetting and eventual recovery. The curriculum progression from single test case mastery to multiple test cases proved effective. **Stage 3** excelled in multi-objective optimization, maintaining high correctness (84.8%) while achieving exceptional efficiency (68.4% optimal programs), with consistent improvement throughout 4000 episodes.

Key insights include: (1) **Abstraction necessity** - Stage 1's limitations highlighted the critical need for appropriate action representation levels, (2) **Training robustness** - Stage 2 revealed the importance of stabilization mechanisms for complex RL tasks, (3) **Multi-objective success** - Stage 3 demonstrated effective balancing of competing objectives, and (4) **Domain knowledge integration** - Pattern recognition significantly accelerated learning efficiency.

The progression from 100% syntax validity $\rightarrow$ 89.5% functional correctness $\rightarrow$ 84.8% correctness with 68.4% optimal efficiency demonstrates successful escalation of program synthesis capabilities through appropriate problem decomposition and curriculum design.

5 Conclusion

The three-stage approach successfully demonstrated that RL can learn increasingly sophisticated program synthesis tasks through careful problem decomposition, appropriate abstraction levels, and domain-informed curriculum design. The key insight was that direct modeling of program semantics (conditional swaps) proved far more tractable than learning low-level syntax generation. While Stage 1 established feasibility, Stages 2-3 achieved meaningful program synthesis with curriculum learning and sophisticated multi-objective optimization. Future work should address the catastrophic forgetting challenges observed in Stage 2 through improved stabilization techniques and further leverage the pattern recognition successes of Stage 3.