A Reasoning-Driven Generative Model for Structured Code Synthesis with Execution-Based Reward Evaluation

Alakh Sharma Gaurish Trivedi

April 9, 2025

Abstract

In this paper, we present an extensively mathematical framework for reasoning-based code synthesis that integrates parameter-efficient fine-tuning via Low-Rank Adaptation (LoRA) with a sampling-based inference mechanism enhanced by an execution-based reward evaluation. We rigorously develop the theoretical foundations of the method, including proofs of correctness, convergence, stability, and complexity bounds. Our approach provides formal guarantees on both functional correctness and resource constraints, and we extend the analysis to cover a wide spectrum of scenarios through detailed lemmas, propositions, and corollaries.

Contents

1 Introduction

Automated code synthesis has become an increasingly prominent research area, leveraging advances in deep learning and formal verification. In this work, we propose a comprehensive framework that combines reasoning-based generative modeling with rigorous mathematical analysis. The system is built upon a transformer model fine-tuned via Low-Rank Adaptation (LoRA) and is augmented by an execution-based reward function. We focus on extending the mathematical rigor by introducing proofs and additional theoretical constructs. Our goal is to derive precise conditions under which the generated code is guaranteed to be functionally correct while satisfying resource constraints.

2 Preliminaries

2.1 Notation and Definitions

Let $\mathcal{P}$ denote the space of problem prompts and $\mathcal{C}$ the space of candidate code solutions. We define:

$$M_{\text{base}} : \mathcal{P} \to \mathbb{R}^d,$$

where d is the hidden dimension of the model. The fine-tuned model M_θ is defined as:

$$M_\theta(x) = M_{\text{base}}(x) + A_\theta x, \quad \forall\, x \in \mathcal{P},$$

with $A_\theta \in \mathbb{R}^{r\times d}$ being a low-rank matrix and $r \ll d$.

Given a dataset:

$$\mathcal{D} = \{(P_i, C_i)\}_{i=1}^N,$$

we minimize the cross-entropy loss:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \text{CE}(M_\theta(P_i), C_i),$$

where $\text{CE}(\cdot,\cdot)$ denotes the cross-entropy function defined on token distributions.

2.2 Modeling the Sampling Process

For an input prompt $P \in \mathcal{P}$, our inference mechanism samples k candidate solutions from the distribution induced by M_θ. Let

$$\Omega_P = \{C^{(1)}, C^{(2)}, \dots, C^{(k)}\} \subset \mathcal{C},$$

where each sample is drawn as

$$C^{(i)} \sim \text{Sample}(M_\theta, P).$$

We assume that the sampling procedure satisfies a probability measure $\mu : \mathcal{C} \to [0,1]$, such that for any measurable subset $A \subseteq \mathcal{C}$,

$$\mu(A) = \int_A p_{M_\theta}(C \mid P)\, dC.$$

The quality of a candidate solution is evaluated via an execution-based reward function detailed in the next section.

3 Execution-Based Reward Function

3.1 Definition of the Reward Function

Let $T = \{(x_j, y_j)\}_{j=1}^n$ be a set of test cases where x_j represents inputs and y_j represents the expected outputs according to the target function f. Define the reward function $R : \mathcal{C} \times T \to \mathbb{R}$ as:

$$R(C,T) = \frac{1}{n}\sum_{j=1}^{n} \delta\left(\text{Exec}(C, x_j) = y_j\right) - \lambda_{\text{err}}\mathbf{1}_{\text{err}}(C) - \lambda_{\text{tle}}\mathbf{1}_{\text{tle}}(C) - \lambda_{\text{mle}}\mathbf{1}_{\text{mle}}(C),$$

where:

- $\delta : \{0,1\} \to \{0,1\}$ is the indicator function.
- $\mathbf{1}_{\text{err}}(C)$, $\mathbf{1}_{\text{tle}}(C)$, and $\mathbf{1}_{\text{mle}}(C)$ are indicator functions for compilation errors, time-limit exceedance, and memory-limit exceedance, respectively.
- $\lambda_{\text{err}}, \lambda_{\text{tle}}, \lambda_{\text{mle}} > 0$ are penalty coefficients.

3.2 Optimal Candidate Selection

The optimal candidate is chosen by maximizing the reward:

$$C^* = \arg\max_{C \in \Omega_P} R(C,T).$$

We assume that the reward function is well-defined and that there exists a candidate $C \in \Omega_P$ for which $R(C,T)$ attains its maximum.

4 Correctness and Resource Guarantees

4.1 Correctness Guarantee

Theorem 4.1 (Correctness)**.** *Let $C \in \mathcal{C}$ be a candidate solution and $T = \{(x_j, y_j)\}_{j=1}^n$ be a comprehensive test set such that $\forall x \in \mathcal{X}$, there exists $(x,y) \in T$ with $y = f(x)$. If $R(C,T) = 1$, then C correctly computes $f(x)$ for all $x \in \mathcal{X}$ under the specified resource constraints.*

Proof. Since $R(C,T) = 1$, it follows that

$$\frac{1}{n}\sum_{j=1}^{n} \delta\left(\text{Exec}(C, x_j) = y_j\right) = 1,$$

implying that for every j, $\text{Exec}(C, x_j) = y_j$. In addition, the penalty terms vanish:

$$\mathbf{1}_{\text{err}}(C) = \mathbf{1}_{\text{tle}}(C) = \mathbf{1}_{\text{mle}}(C) = 0.$$

Thus, C produces the correct output for every test case, and given that T spans the domain $\mathcal{X}$, the candidate C correctly computes $f(x)$ for all $x \in \mathcal{X}$. □

4.2 Resource Constraint Formalization

Define the resource-constrained set:

$$\mathcal{C}_{t,m} = \{C \in \mathcal{C} \mid \text{ExecTime}(C) \leq t,\ \text{MemUsage}(C) \leq m\}.$$

Let $\mathcal{C}_{\text{valid}} \subset \mathcal{C}$ denote the set of syntactically valid and compilable programs. We require that:

$$C^* \in \mathcal{C}_{t,m} \cap \mathcal{C}_{\text{valid}},$$

ensuring that the chosen solution is not only correct but also feasible with respect to resource limitations.

5 Extended Theoretical Analysis

5.1 Lipschitz Continuity of the Model

Definition 5.1 (Lipschitz Continuity). *A function $F : \mathcal{X} \to \mathcal{Y}$ is said to be Lipschitz continuous if there exists a constant $L > 0$ such that:*

$$\|F(x_1) - F(x_2)\| \leq L\|x_1 - x_2\|, \quad \forall x_1, x_2 \in \mathcal{X}.$$

Lemma 5.2 (Lipschitz Property of M_θ). *Assume M_{base} is Lipschitz continuous with constant L_0, and the low-rank adaptation A_θ satisfies $\|A_\theta\| \leq L_A$. Then, for any $x_1, x_2 \in \mathcal{P}$,*

$$\|M_\theta(x_1) - M_\theta(x_2)\| \leq (L_0 + L_A)\|x_1 - x_2\|.$$

Proof. We have

$$\begin{aligned} \|M_\theta(x_1) - M_\theta(x_2)\| &= \|M_{\text{base}}(x_1) - M_{\text{base}}(x_2) + A_\theta(x_1 - x_2)\| \\ &\leq \|M_{\text{base}}(x_1) - M_{\text{base}}(x_2)\| + \|A_\theta(x_1 - x_2)\| \\ &\leq L_0\|x_1 - x_2\| + L_A\|x_1 - x_2\| \\ &= (L_0 + L_A)\|x_1 - x_2\|. \end{aligned}$$

□

5.2 Convergence of the Sampling Process

Proposition 5.3 (Convergence of Maximal Reward). *Let $\{C^{(i)}\}_{i=1}^k$ be candidate solutions generated by the sampling process, with rewards $\{R(C^{(i)}, T)\}_{i=1}^k$. Under the assumption that there is a non-zero probability $p > 0$ of sampling a candidate C such that $R(C, T) = 1$, we have*

$$\lim_{k\to\infty} \mathbb{P}\left(\max_{1\leq i\leq k} R(C^{(i)}, T) = 1\right) = 1.$$

Proof. Let E_i be the event that $R(C^{(i)}, T) = 1$. Then $\mathbb{P}(E_i) = p$. The events E_i are i.i.d. Thus,

$$\mathbb{P}\left(\bigcup_{i=1}^k E_i\right) = 1 - (1-p)^k.$$

Taking the limit as $k \to \infty$, we have

$$\lim_{k\to\infty} \mathbb{P}\left(\bigcup_{i=1}^k E_i\right) = 1.$$

Thus, the maximal reward converges to 1 with probability 1. □

5.3 Error Bounds in Reward Estimation

Let $\widehat{R}(C, T)$ be an empirical estimate of $R(C, T)$ based on a finite number n of test cases. By Hoeffding's inequality, for any $\epsilon > 0$,

$$\mathbb{P}\left(\left|\widehat{R}(C, T) - R(C, T)\right| \geq \epsilon\right) \leq 2\exp\left(-2n\epsilon^2\right).$$

This establishes the statistical reliability of the reward estimation as n increases.

6 Algorithmic Framework

6.1 Inference Pipeline

The following algorithm summarizes the full inference process:

Algorithm 1 Extended Inference Pipeline for Code Synthesis

```
Require: Fine-tuned model $M_\theta$, problem prompt $P$, test set $T$, number of samples $k$
1: Generate candidates: $\Omega_P = \{C^{(1)}, \ldots, C^{(k)}\} \leftarrow \text{Sample}(M_\theta, P)$
2: for $i = 1$ to $k$ do
3:     $C^{(i)} \leftarrow \text{ParseAndSanitize}(C^{(i)})$
4:     Compute reward: $r^{(i)} \leftarrow R(C^{(i)}, T)$
5: end for
6: return $C^* \leftarrow \arg\max_{1 \leq i \leq k} r^{(i)}$
```

6.2 Complexity Analysis

Assume that the time complexity to parse and sanitize a candidate is $O(p)$, and the execution of the reward function for one candidate is $O(q)$, then the overall time complexity for k candidates is:

$$O\left(k(p+q)\right).$$

Moreover, if the model's inference has complexity $O(m)$ per sample, then the total complexity becomes:

$$O\left(k(m+p+q)\right).$$

This analysis allows us to balance the trade-off between sample size and computational resources.

7 Extensions and Further Mathematical Considerations

7.1 Nonlinear Generalizations

While our model M_θ is linear in its adaptation, one can consider a more general nonlinear adaptation:

$$M_\theta(x) = M_{\text{base}}(x) + \phi_\theta(x),$$

where $\phi_\theta : \mathcal{P} \to \mathbb{R}^d$ is a nonlinear function parameterized by θ. Under mild smoothness conditions on ϕ_θ, similar Lipschitz continuity arguments can be made, albeit with more complex constants.

7.2 Stability Analysis under Perturbations

Let $P \in \mathcal{P}$ be a prompt and consider a perturbation ΔP. We analyze the change in the model's output:

$$\|M_\theta(P + \Delta P) - M_\theta(P)\| \leq L\|\Delta P\|,$$

where L is the Lipschitz constant derived earlier. This stability result is crucial for ensuring robustness in the face of noisy inputs.

7.3 Extended Proofs for Resource Constraints

Consider a candidate $C \in \mathcal{C}_{t,m}$ with the property that

$$\text{ExecTime}(C) \leq t \quad \text{and} \quad \text{MemUsage}(C) \leq m.$$

Define the resource utilization function $\rho : \mathcal{C} \to \mathbb{R}^2$ by

$$\rho(C) = (\text{ExecTime}(C), \text{MemUsage}(C)).$$

Then, the set $\mathcal{C}_{t,m}$ can be written as

$$\mathcal{C}_{t,m} = \{C \in \mathcal{C} \mid \rho(C) \preceq (t, m)\}.$$

Using standard techniques in convex analysis, if ρ is convex, then $\mathcal{C}_{t,m}$ is a convex set. This enables the use of projection methods to ensure that C^* satisfies resource constraints optimally.

8 Conclusion and Future Work

We have developed an extensively mathematical framework for reasoning-based code synthesis. By incorporating detailed proofs of correctness, convergence, stability, and complexity bounds, our framework not only ensures functional correctness under resource constraints but also provides a rich theoretical basis for further research. Future work will focus on extending these methods to more complex non-linear models, integrating formal verification techniques, and optimizing the trade-offs between sampling efficiency and computational complexity.