NeuroSymbolic LLM for Mathematical Reasoning and Software Engineering

Prithwish Jana
School of Computer Science, Georgia Institute of Technology, Atlanta, USA
pjana7@gatech.edu

Abstract

In recent years, there has been a significant interest in Large Language Models (LLMs) owing to their notable performance in natural language processing (NLP) tasks. However, while their results show promise in mathematical reasoning and software engineering tasks, LLMs have not yet achieved a satisfactory performance level in these domains. In response, current approaches have prioritized scaling up the size of LLMs, necessitating substantial computational resources and data. Our objective, however, is to pursue a different path by developing neurosymbolic language models. We propose to integrate logical and symbolic feedback during the training process, enabling significantly smaller language models to achieve far better reasoning capabilities than the LLMs currently in use.

1 Introduction

Artificial Intelligence (AI) deals with the ability of machines to engage in cognitive tasks that involve human-like thinking and problem-solving abilities. Under the broad field of AI, machine learning (ML) and logical/automated reasoning (AR) are traditionally viewed as two distinct and orthogonal approaches. While ML (e.g., neural net, LLM) enables computers to estimate patterns from data, AR (e.g., satisfiability or SAT solver, satisfiability modulo theories or SMT solver, computer algebra system or CAS) involves using structured rules and formal reasoning to make decisions and draw conclusions. In recent years, LLMs have made a substantial impact across diverse AI sub-fields. However, their performance is inconsistent, excelling in *natural language tasks* like sentiment analysis or text summarization, yet encountering challenges in *formal language tasks* requiring deductive or inferential reasoning e.g., theorem proving, code generation.

To capitalize on the language understanding strengths of the LLMs and incorporate the logical reasoning capabilities necessary for formal languages, we propose to integrate solvers (AR engines) into the LLM training process. These solvers work in tandem with the LLMs to form a corrective feedback loop [Ganesh *et al.*, 2022], where the LLM constructs initial abstractions, and an AR engine verifies and corrects them. By leveraging this iterative approach, we aim to address classes of AI problems that would otherwise be exceptionally challenging or even infeasible to solve. While building efficient neurosymbolic LLMs for such problems, the research questions (RQs) that we intend to address are:

- **RQ1:** Are LLMs good in reasoning tasks? Can they be significantly improved by providing logical feedback about the quality of their generated results, during training?
- **RQ2:** How do we design feedback loops to train LLMs for software engineering and mathematical reasoning tasks?
- **RQ3:** What type of feedback are the most effectively received by LLMs: scalar, vector or multi-dimensional?
- **RQ4:** Can a small language model (SLM) trained using symbolic feedback, match the performance of larger models (LLMs) trained without such feedback? Can we theoretically prove them to be more trustworthy than the latter?

Overall, we aim to answer the following: *Can LLMs being trained for software engineering or mathematical reasoning tasks benefit from feedback from symbolic engines or solvers?*

2 Problem Formulation

In this paper, we focus on developing neurosymbolic NLP systems for code translation in the field of *software engineering* and theorem proving in *mathematical reasoning*.

LLM for Code Translation. The automated translation of entire programs from one high-level programming language to another (e.g., from Java to Python) is a significant field of research in software engineering. This area has practical applications in code migration and ensuring cross-platform interoperability. To formally elaborate the problem setting, let S denote a source language and T be a target language. The *code translation learning problem* is to train a language translator $f_{ST} : S \rightarrow T$, where given a S-program the translator produces a T-program that (a) adheres to the grammar of T and (b) maintains functional (input-output) equivalence with the original S-program w.r.t. a given test-suite.

LLM for Theorem Proving. The International Mathematics Olympiad (IMO) problems necessitate intricate mathematical reasoning and demand thorough proof from participants as a solution. When attempted with an LLM, the task can be formally defined as follows: Given a statement in natural language (NL), generate a proof in NL such that the formal language (FL) counterpart of the proof can be verified. This involves three aspects: (a) *auto-formalization* i.e., translating an NL statement into a syntactically-correct FL statement (in languages like Lean [Moura and Ullrich, 2021]), (b) *theorem proving* i.e., producing an FL proof for the FL statement and (c) *auto-informalization* i.e., translating the FL proof back to NL for easy interpretation by humans.

3 Preliminary Contributions

We have built an LLM-based code translation tool named CoTran [Jana *et al.*, 2024], through which we have partially addressed the RQs. We are also in the process of developing a neurosymbolic LLM to solve IMO-level math problems, which will deepen our investigation into the RQs.

CoTran. Handcrafted rule-based source-to-source compilers or transpilers are expensive and tedious to build. This has led to the development of advanced code LLMs e.g., CodeBERT, CodeGPT, CodeT5. We observed that encoder-decoder transformer-based LLMs when trained using supervised fine-tuning (SFT) to minimize cross-entropy loss, performs much better than rule-based transpilers. Having said that, the code translation performance improves dramatically upon incorporating logical feedback signals during training.

The contributions in CoTran are three-fold. Firstly, we introduce a *keyword tokenizer (kw-Tok)*, that adapts any existing tokenizer like RoBERTa for programming languages. This is done by adding all the keywords and operators of languages S and T, to the tokenizer vocabulary. Secondly, we introduce *Compiler Feedback (CF)* and *Symbolic-Execution Feedback (SF)*, which are scalar fine-grained reward functions. CF is computed by the T-compiler, which identifies the relative position of the first compilation error. It assists an LLM during fine-tuning by helping it assess how close the generated translations are to a perfectly compiling one. SF is used to fine-tune $S \rightarrow T \rightarrow S$ back-to-back LLMs. We perform automated unit-test generation for the input code and evaluate them on the translated code to provide feedback on any inequivalence. Thirdly, we *interleave* reinforcement learning (RL)-based fine-tuning by Proximal Policy Optimization and SFT-based optimization of cross-entropy loss to incorporate logical feedback during LLM training. This prevents LLMs from deviating from the cross-entropy loss objective.

CoTran, built on CodeT5-base, is extensively evaluated on Java-to-Python (J2P) and Python-to-Java (P2J) translation against 11 state-of-the-art LLM-based tools and 3 human-written transpilers. Performance is measured using metrics like functional equivalence accuracy (FEqAcc), compilation accuracy (CompAcc), BLEU, and CodeBLEU. For J2P and P2J, CoTran improves FEqAcc by +12.94% and +14.89%, and CompAcc by +4.30% and +8.14%, respectively (**RQ1**). Further, interleaving RL and SFT improves the LLM's performance compared to using RL alone (**RQ2**). With the fine-grained CF and SF, CoTran achieves +11.57% higher FEqAcc and +17.36% higher CompAcc for P2J translation than using Boolean feedback in an RL setup (**RQ3**). CoTran, built on the CodeT5-base model with $\sim$220M parameters, is compared to ChatGPT (GPT-3.5), which has over 100B parameters. For P2J translation, CoTran outperforms ChatGPT by +27.03% in FEqAcc and +52.01% in CompAcc. This shows that a significantly smaller language model can surpass a larger one when logical feedback is incorporated during training (**RQ4**).

NeuroSymbolic Theorem Prover. AlphaGeometry [Trinh *et al.*, 2024] is an LLM-based neurosymbolic theorem prover for Euclidean plane geometry that performs at the level of an average IMO gold medalist. We aim to develop a smaller language model trained with feedback from proof assistants and verifiers, that can outperform much larger LLMs in specialized settings. The advantage of such a system is that, when the proof of a theorem is expressed in FLs like Lean, it can undergo formal verification by Lean's proof-checking system to ensure its correctness. This process helps establish a high confidence in the correctness of the generated proofs.

In this regard, the auto-formalization and theorem-proving steps in Section 2 can significantly benefit from AR feedback by an automated theorem prover (ATP). An ATP can be used to generate a certificate detailing the faults (e.g., which proof blocks or open conjecture could not be proven) in the LLM-generated formal statement and proof. When integrated as feedback within an RL framework, this certificate has the potential to effectively train the LLM towards proof generation.

4 Conclusion and Future Work

Overall, we intend to develop specialized small language models for tasks in software engineering (e.g. code generation from NL description) and mathematical reasoning (e.g. inferring the truth value of an NL question from NL facts + rules), that can excel through a fusion of ML and automated reasoning. As a next step, we aim to explore vector and multi-dimensional feedback in the RL setting, as opposed to scalar feedback. Through an in-depth theoretical study, we aim to determine the conditions for which an LLM trained with feedback is likely to outperform one trained without feedback.

Acknowledgements

I would like to express my sincerest gratitude to my Ph.D advisor, Prof. Vijay Ganesh. His support and leadership have been instrumental in making this work possible.

References

[Ganesh *et al.*, 2022] Vijay Ganesh, Sanjit A Seshia, and Somesh Jha. Machine Learning and Logic: A New Frontier in Artificial Intelligence. *Formal Methods in System Design*, 60(3):426–451, 2022.

[Jana *et al.*, 2024] Prithwish Jana, Piyush Jha, Haoyang Ju, Gautham Kishore, Aryan Mahajan, and Vijay Ganesh. CoTran: An LLM-based Code Translator using Reinforcement Learning with Feedback from Compiler and Symbolic Execution. *arXiv:2306.06755*, 2024.

[Moura and Ullrich, 2021] Leonardo de Moura and Sebastian Ullrich. The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction–CADE 28*, pages 625–635. Springer, 2021.

[Trinh *et al.*, 2024] Trieu Trinh, Yuhuai Wu, Quoc Le, He He, and Thang Luong. Solving Olympiad Geometry without Human Demonstrations. *Nature*, 625, 2024.