High Confidence Software and Systems (HCSS) Conference

March 12, 2025

Neuro-Symbolic Techniques for LLM-based Code Generation, Translation and Auto-Formalization

Prithwish Jana & Vijay Ganesh

School of Computer Science,
Georgia Tech, Atlanta, USA

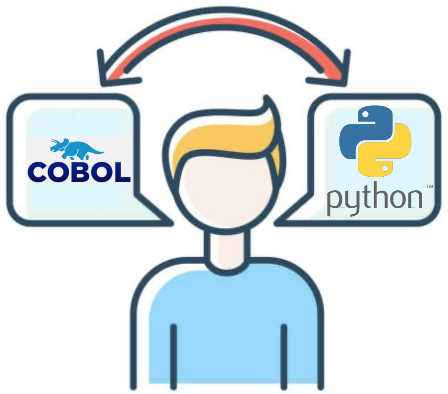
: pjana7@gatech.edu, vganesh@gatech.edu

Georgia Tech

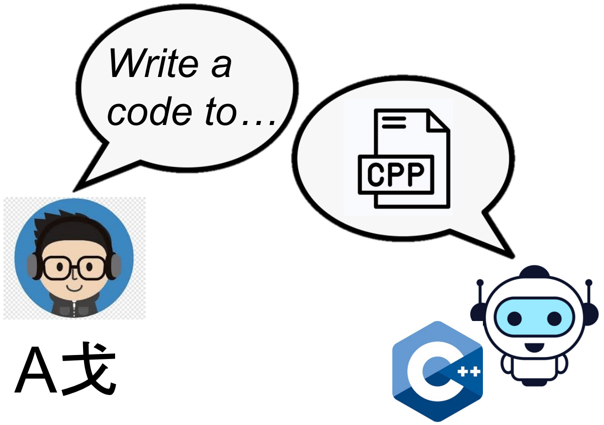
INTRODUCTION

Motivation: AI for Software Engineering (SE)

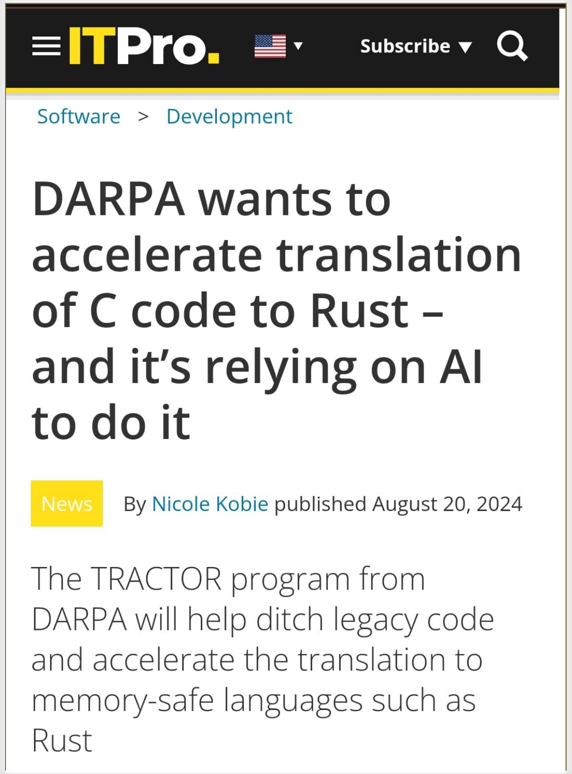
Code Translation (from one high-level language to another)



Code Generation (natural language to code)



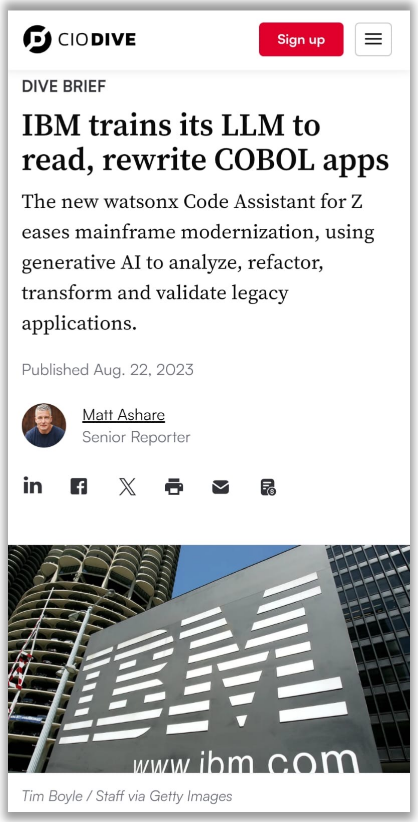
Code Repair, etc.


Software > Development

DARPA wants to accelerate translation of C code to Rust – and it's relying on AI to do it

News By Nicole Kobie published August 20, 2024

The TRACTOR program from DARPA will help ditch legacy code and accelerate the translation to memory-safe languages such as Rust


DIVE BRIEF

IBM trains its LLM to read, rewrite COBOL apps

The new watsonx Code Assistant for Z eases mainframe modernization, using generative AI to analyze, refactor, transform and validate legacy applications.

Published Aug. 22, 2023

Matt Ashare
Senior Reporter

Tim Boyle / Staff via Getty Images

- AI is already being used to automate various SE tasks:

code llama | Gemini Code Assist | GitHub Copilot | CURSOR The AI Code Editor | ChatGPT | Claude 3.7 | deepseek coder


Google's Duet AI targets legacy software revolution

News By Ross Kelly published August 30, 2023

Google is hoping to eliminate the pain involved in refactoring code in legacy software into more modern programming languages


FRUIT DECK

Apple has a Proton-like Game Porting Toolkit for getting Windows games on Mac

Eager gamers already have *Cyberpunk*, *Diablo IV* running on Apple Silicon Macs.

KEVIN PURDY - 7 JUN 2023 12:29

Motivation: AI for Mathematical Reasoning

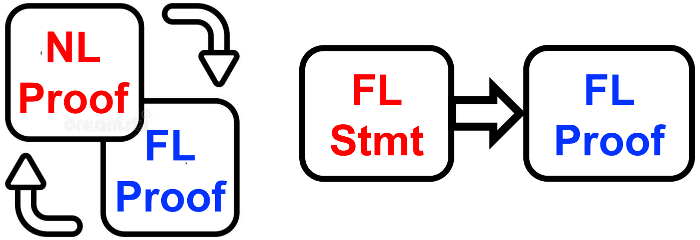
- **Computer-Verifiable Proofs: rigorous, machine-checkable verification without ambiguity**
  - Formal mathematical languages facilitate *computer-verifiable proofs*
    E.g., Lean4, Peano, Metamath, HOL Light, Isabelle, Coq
  - Formal proof vs. Natural Language proof
    - Uses strict syntactic rules and symbolic logic

Metamath

Isabelle

- **Challenges of Writing Proofs in a Formal Language (FL)**
  - Formalizing proofs requires significant time, can be difficult even for experienced mathematicians

- **Emerging research to simplify writing proofs in FL**
  - **Auto-Formalization**
    - Translates natural language (NL) proofs into formal language (FL) proofs
  - **Automated Formal Proof Synthesis (aFPS)**
    - Generates formal proofs directly from statements (conjectures) in FL



- **AI being used to automate various formalization tasks:**
  - Reduces human effort
  - Enables non-experts to use formal methods
  - Discover new theorems beyond intuition

DeepSeek @deepseek_ai
DeepSeek-Prover 2

DeepMind
AlphaGeometry

Goedel-LM
/Goedel-Prover-SFT

lean-dojo/
ReProver

FrenzyMath
/Herald_translator

Lean-STaR

Georgia Tech

What's Common? Formal Language Tasks

- **Formal Language (FL)**
  - Well-formed, strict & unambiguous; specific set of rules (formal grammar)
  - Designed for applications in maths, computer science, and logic
  - E.g., Programming languages (C++, Java, Python),
    First- or higher-order logic (Lean, Isabelle, Coq)

A Python program to check prime number

```python
num = int(input("Please input a number:"))

flag = False
if num == 0 or num == 1:
    print(num, "is not a prime number")
elif num > 1:
    for i in range(2, int(num**0.5) + 1):
        if (num % i) == 0:
            flag = True
            break

    if flag:
        print(num, "is not a prime number")
    else:
        print(num, "is a prime number")
```

```lean
theorem transitive (x y z: Nat)
  (h1 : x = y) (h2 : y = z) :
  x = z := by
rewrite [h1]
rewrite [h2]
rfl
```

A Lean4 proof for transitive property of equality for natural numbers

ChatGPT, MISTRAL AI_, Gemma 2, code llama, deepseek coder, ...

Issues faced when prompting LLMs to generate codes/proofs

- Syntactic/semantic errors
- Wrong API (unavailable APIs or hallucinate with similar APIs)
- Introduce *'subtle bugs'* in the code
- *Overall, LLMs unable to abide by strict rules of a FL*

Objective: LLMs for Formal Languages

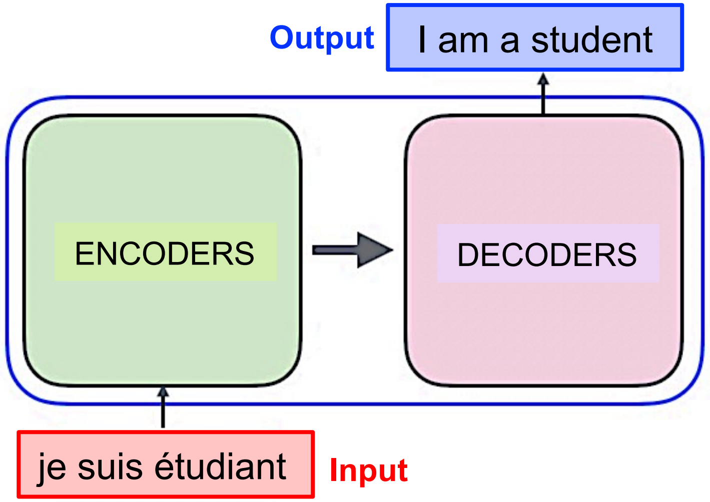
LLMs strong in language understanding

- LLMs excel in **natural language tasks**
  - Sentiment analysis, Text summarization, etc.
  - **Approximate results** oftentimes **good enough**
    E.g., "**I am a student**" and "**I am student**" both sound fine

LLMs weak in adhering to syntax, performing logical reasoning

- LLMs face challenges in **formal language tasks**
  - **Formal grammar:** well-formed, strict, unambiguous
  - Tasks like **code translation**, **theorem proving**, etc.
  - SoTA prioritized scaling-up LLM size $\Longrightarrow$ data $\uparrow$, resources $\uparrow$

Not a sustainable solution

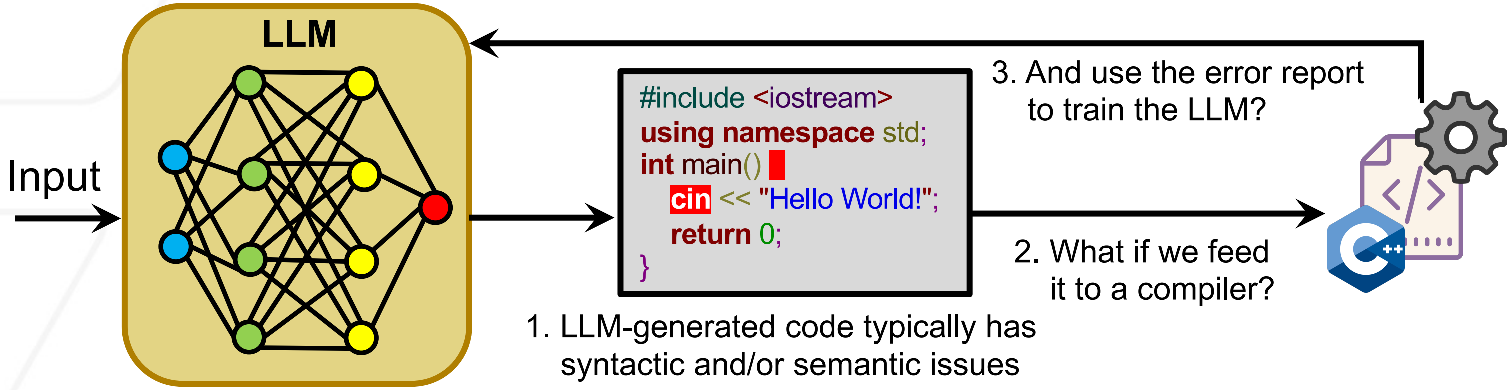


Transformer[1] architectures (LLMs) trained on next-token prediction over large text corpora

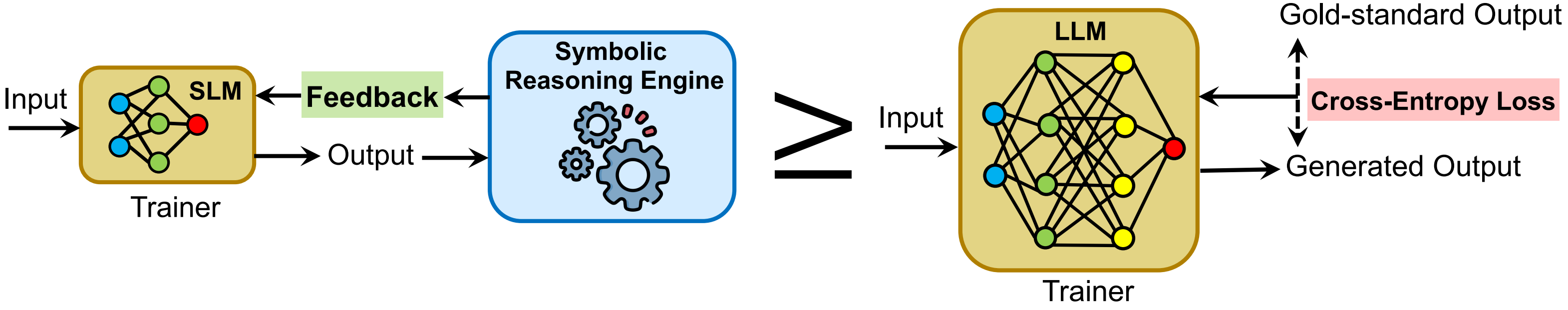
[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is All You Need. *Advances in Neural Information Processing Systems, 30*.

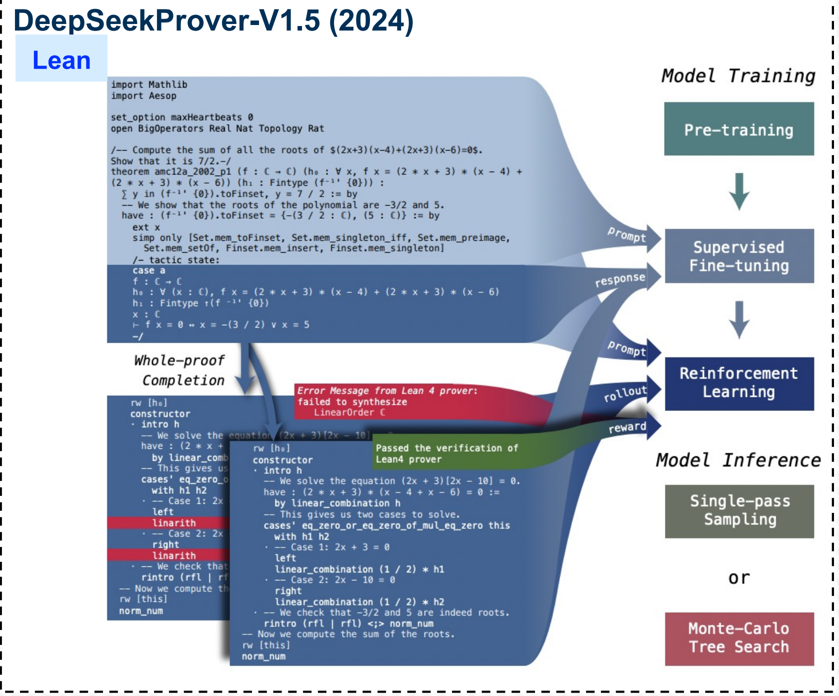
Research Questions (RQs)

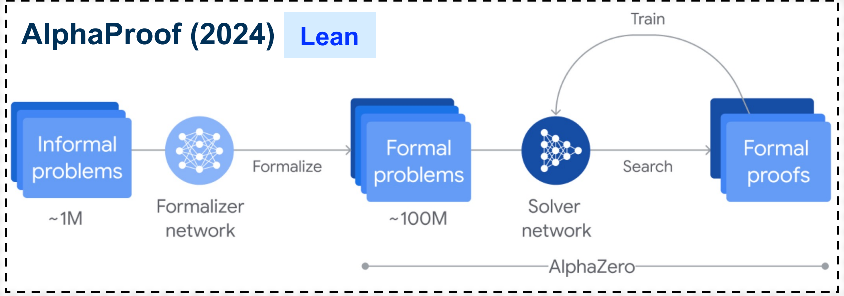
➢ **RQ-1:** Can LLMs be improved in reasoning tasks by providing ***fine-grained logical feedback*** during fine-tuning?



➢ **RQ-2:** Can a ***small language model (SLM)*** trained using symbolic feedback, outperform LLMs without feedback?







Researchers are already **combining LLMs and formal verification tools**

- e.g., DeepSeek-Prover uses **binary feedback** in an RL loop **(sparse reward!)**

Georgia Tech

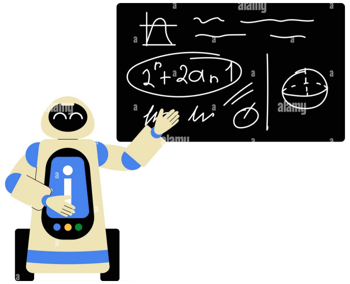
REST OF THE TALK

- RLSF: Reinforcement Learning via Symbolic Feedback (under submission)

- CoTran: An LLM-based Code Translator using RL with Feedback from Compiler & Symbolic Exec. **(ECAI-2024)**

AI for Software Engineering

- Automated Proof Synthesis and Auto-Formalization using LLMs + LEAN



AI for Mathematics

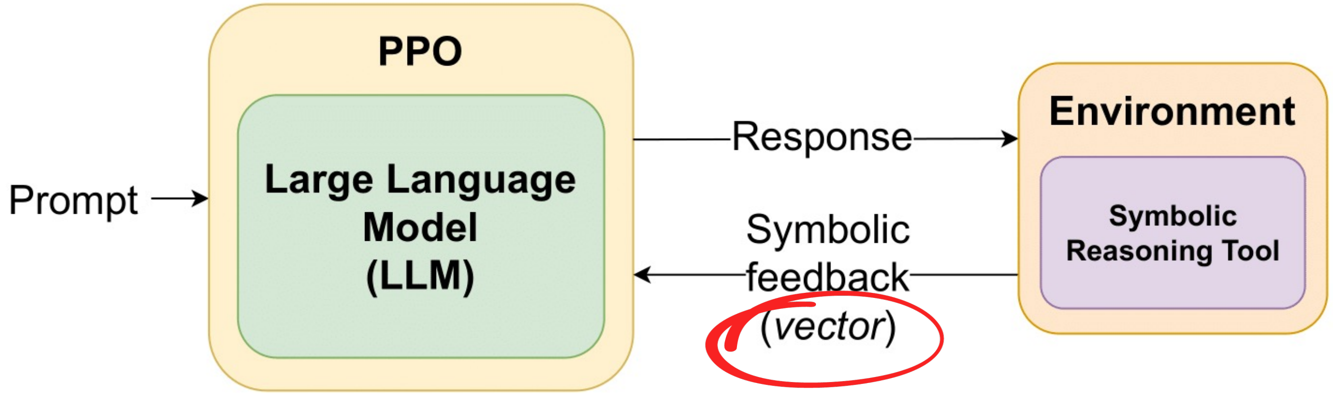
- NeuroSymbolic LLM for Mathematical Reasoning & Software Engg. **(IJCAI-2024)**

RLSF: Reinforcement Learning via Symbolic Feedback

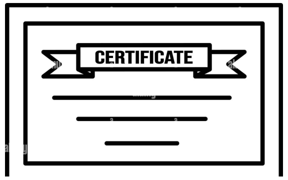
Piyush Jha, Prithwish Jana, Pranavkrishna Suresh,
Arnav Arora, Vijay Ganesh

Georgia Institute of Technology, USA

REINFORCEMENT LEARNING VIA SYMBOLIC FEEDBACK (RLSF) OVERVIEW



- **New fine-tuning paradigm** for LLMs
- LLM acts as the **RL agent**
- The environment is enhanced with **sound symbolic tools** (e.g., solvers, provers) for accurate feedback



- **Symbolic reasoning tools** generate **poly-sized certificates** specifying errors.
- Certificates transformed to **Fine-Grained (Vectorized) Token-Level Feedback**, rather than relying on sparse scalar rewards.

We demonstrated success of RLSF in challenging tasks like **program synthesis**, **molecule generation**, and **mathematical problem-solving**.

Georgia Tech

What others have Tried?

- **Supervised Fine-Tuning:**
  - Using cross-entropy loss to fine-tune LLMs for specific tasks, requires differentiable loss.
- **RLHF Approaches:**
  - Using human-generated feedback to align LLM to preferred responses.
    - **Black-Box Reward Models:** Existing RLHF methods use ***unsound*** models → fail to capture the nuances of reasoning tasks.
    - **Sparse Scalar Rewards:** Difficulty in obtaining detailed feedback.
    - **Data Collection:** Challenges in collecting large-scale, high-quality preference data for fine-tuning.
- **Neuro-symbolic AI:**
  - Integrations of NNs with symbolic reasoning tools, typically within RL agent rather than environment

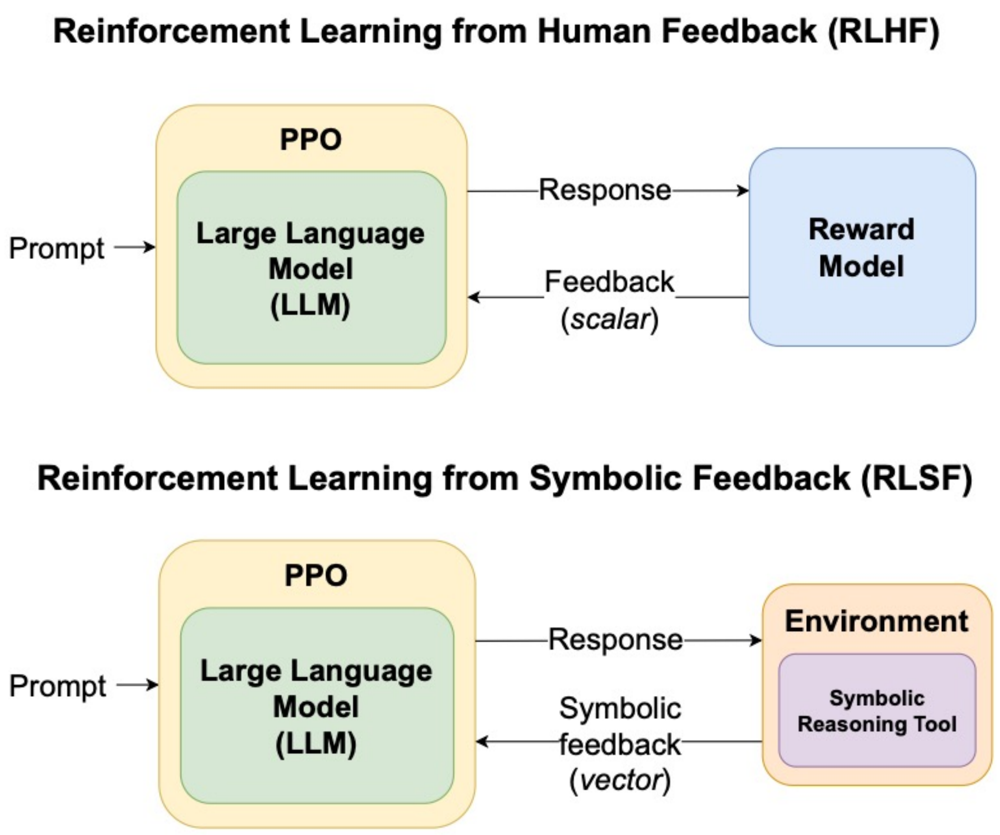


Figure 1: **Contrasting RLHF with RLSF:** The image depicts two distinct fine-tuning paradigms. (Left) RLHF operates within an environment governed by a black-box reward model, typically offering scalar feedback. (Right) By contrast, the environment in RLSF leverages sound symbolic reasoning tools and also provides fine-grained token-level vector feedback that is, in turn, based on poly-sized certificates produced by these symbolic tools.

RLSF ALGORITHM

Algorithm 1 Reinforcement Learning via Symbolic Feedback (RLSF)

Input: Number of epochs N_{epochs}, pre-trained model $Model$, symbolic reasoning tool $SymbolicReasoner$, reward function $RewardFunc$, prompt dataset D

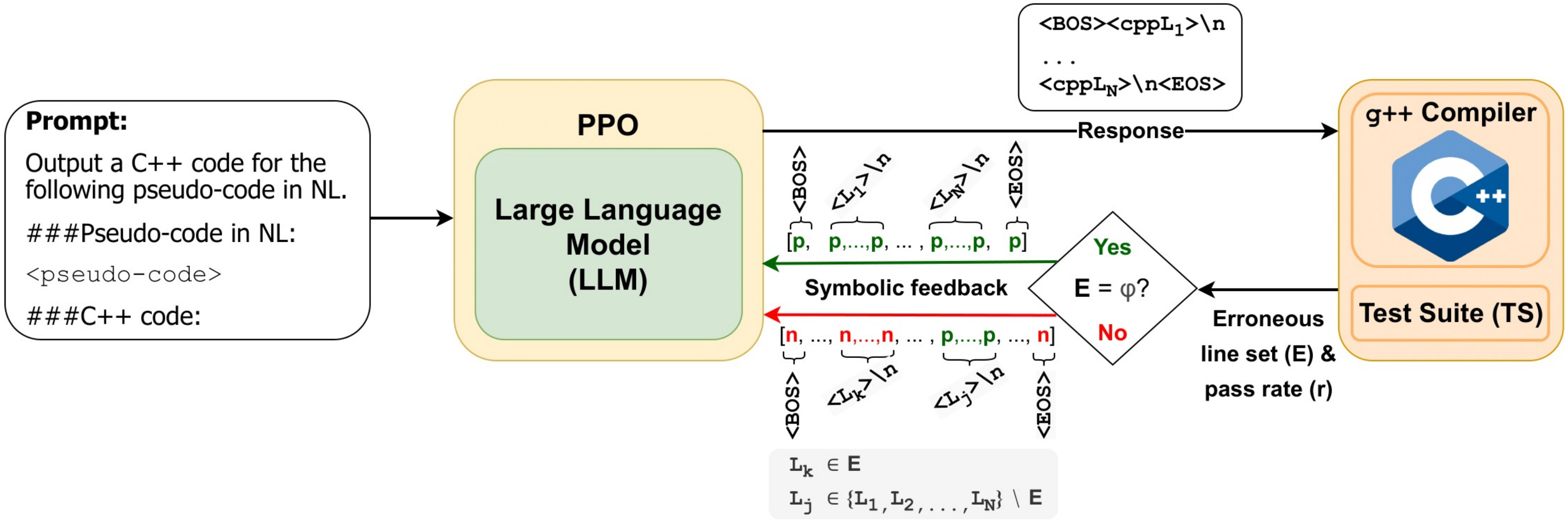
Output: Fine-tuned model $Model'$

```
for epoch in 1, 2, ..., N_epochs do
    for batch_i in D do
        response_i ~ Model(batch_i)
        cert_i ← SymbolicReasoner(batch_i, response_i)
        vector_i ← RewardFunc(cert_i)
        Model' ← ppo_step(Model, batch_i, response_i, vector_i)
        Model ← Model'
    end for
end for
```

I. RLSF for Code Generation: LLM with Verifier Feedback

Code Generation Learning Problem

Learn $f: \boldsymbol{NL} \rightarrow \boldsymbol{PL}$, which when provided with a $\boldsymbol{NL}$ pseudo-code produces a $\boldsymbol{PL}$-program

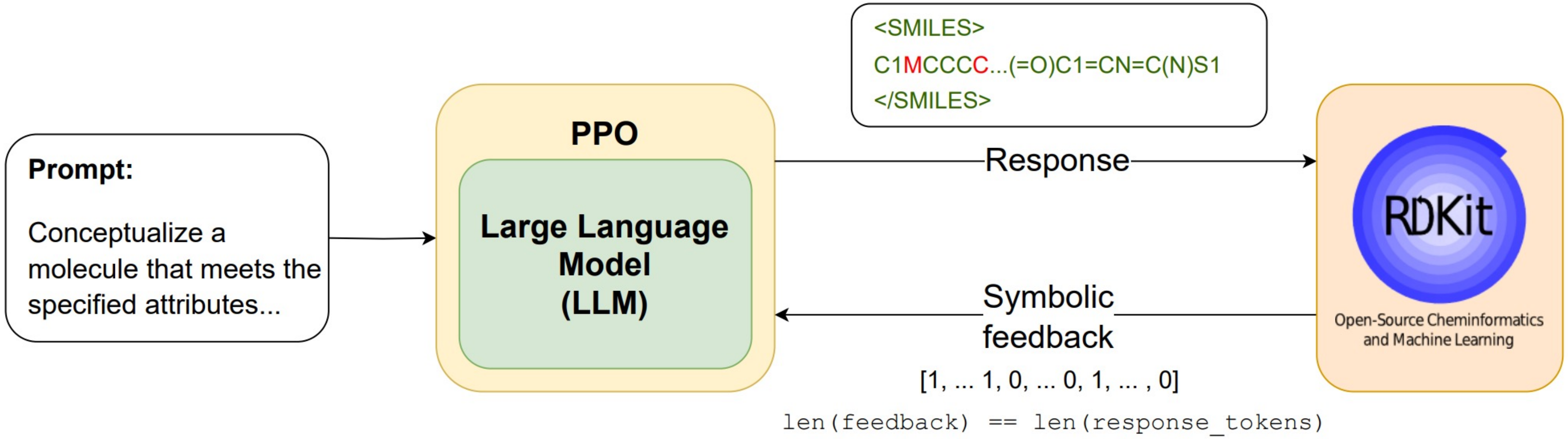


- Given a generated C++ code (with **N** lines), the symbolic environment uses the g++ compiler to detect erroneous lines (**E**) and compute pass rate **r**
- Accordingly, the environment provides fine-grained symbolic feedback for fine-tuning the LLM

II. RLSF FOR CHEMISTRY: LLM WITH DOMAIN KNOWLEDGE OF CHEMISTRY

- **Task 1: Molecule Generation**:
  - Generating chemical structures from natural language descriptions.
- **Task 2: Forward Synthesis**:
  - Predicting the product of chemical reactions given reactants.
- **Task 3: Retrosynthesis**:
  - Identifying reactants required to produce a specific molecule.

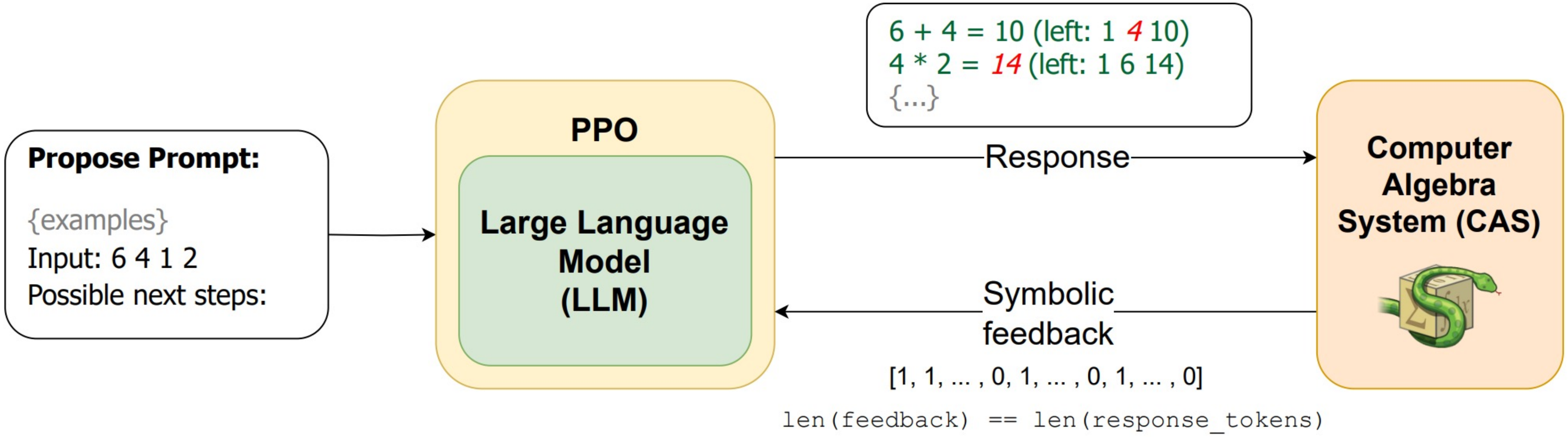
- **Feedback Mechanism**:
  - Uses RDKit to identify syntax errors (e.g., invalid molecule strings) and applies the first law of chemistry for semantic validation.



III. RLSF for Logic Puzzles: LLM with Computer Algebra Systems (CAS)

- **Game of 24:**
  - Solving a mathematical puzzle by finding a sequence of operations to reach the number 24 from four given numbers.

- **Feedback Mechanism**:
  - Uses symbolic math tools (e.g., Computer Algebra System i.e., CAS) to verify the correctness of solutions, providing precise feedback for token-level corrections.



Experimental Results: RLSF for Code Generation

ChatGPT

CodeGemma

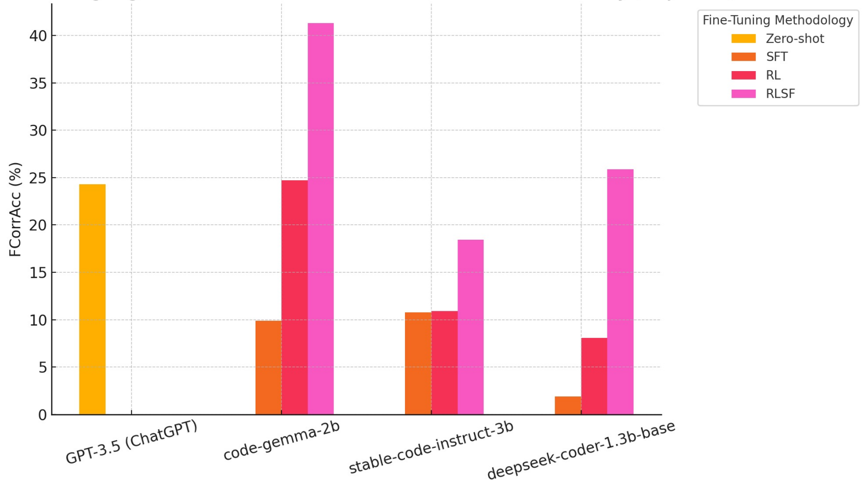
stability.ai

deepseek coder

LLM	Configuration	CompAcc (%)	FCorrAcc (%)	Pass@1 (%)
GPT-3.5 (Achiam et al., 2023)	Zero-shot (no fine-tuning)	**29.13**	**24.29**	**20.98**
code-gemma-2b (CodeGemma-Team, 2024)	Zero-shot (no fine-tuning)	0.00	0.00	0.00
	SFT (with cross-entropy loss)	11.31	9.87	9.00
	SFT + RL (with Boolean scalar f/b)	54.89	24.71	12.77
	SFT + *RLSF (with token-level f/b)*	**63.95**	**41.30**	**28.80**
stable-code-instruct-3b (Pinnaparaju et al., 2024)	Zero-shot (no fine-tuning)	0.00	0.00	0.00
	SFT (with cross-entropy loss)	12.04	10.78	9.96
	SFT + RL (with Boolean scalar f/b)	48.43	10.91	9.22
	SFT + *RLSF (with token-level f/b)*	**54.27**	**18.44**	**16.09**
deepseek-coder-1.3b-base (Guo et al., 2024)	Zero-shot (no fine-tuning)	0.00	0.00	0.00
	SFT (with cross-entropy loss)	2.19	1.90	1.63
	SFT + RL (with Boolean scalar f/b)	19.97	8.07	3.88
	SFT + *RLSF (with token-level f/b)*	**38.92**	**25.89**	**14.51**

- Compared to GPT-3.5, RLSF-tuned **Google's CodeGemma-2b (a 100× smaller model)**
  - improved compilation accuracy by +34.82%
  - improved functional correctness by +17.01%

Natural Language Pseudo-code to Code Translation: FCorrAcc Only (Adjusted)



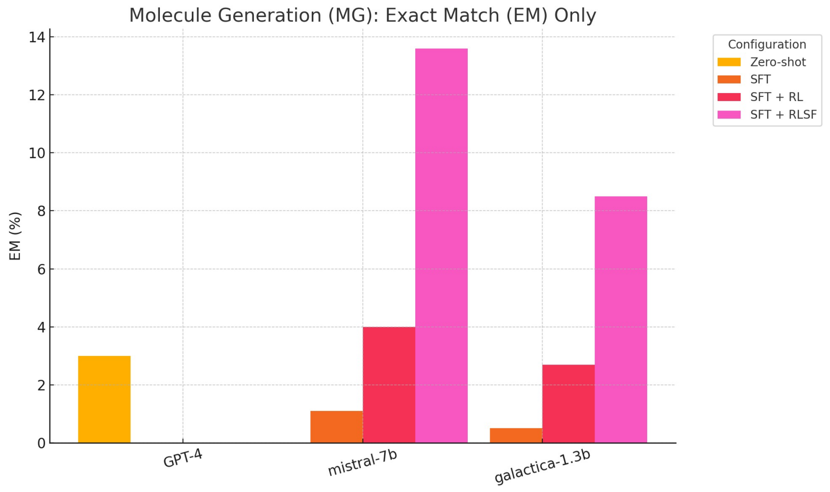
RLSF: Reinforcement Learning via Symbolic Feedback
P. Jha, P. Jana, P. Suresh, A. Arora, V. Ganesh

Georgia Tech

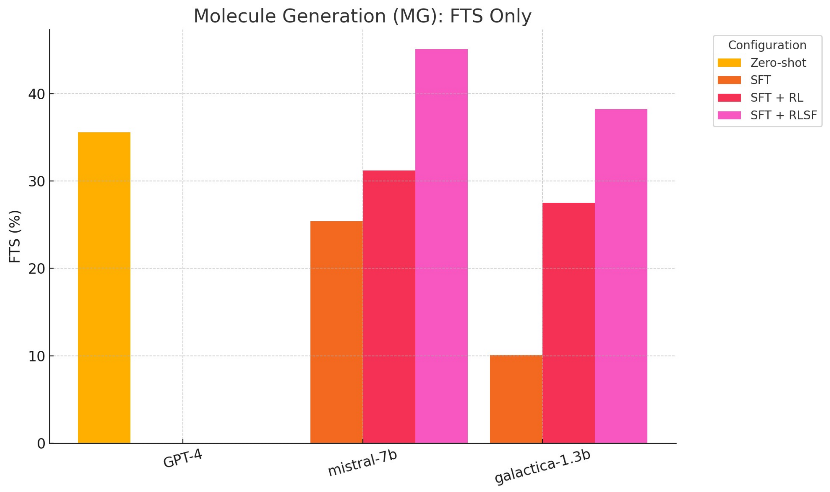
Experimental Results: RLSF for Chemistry

Table 2: **Chemistry Tasks Results:** Performance comparison over different LLMs for three chemistry tasks - Molecule Generation (MG), Forward synthesis (FS), and Retrosynthesis (RS)

LLM	Configuration	Task & Metric: Molecule Generation (MG) EM (%)	MG FTS (%)	MG Valid (%)	Forward synthesis (FS) EM (%)	FS FTS (%)	FS Valid (%)	Retrosynthesis (RS) EM (%)	RS FTS (%)	RS Valid (%)
`GPT-4` (Achiam et al., 2023)	Zero-shot	**3.0**	**35.6**	**90.0**	**0.4**	**37.5**	**93.1**	**0.8**	**31.7**	**88.2**
`mistral-7b` (Jiang et al., 2023)	Zero-shot	0.0	0.0	0.0	0.0	49.8	14.4	0.0	46.4	13.5
	SFT	1.1	25.4	51.9	7.2	54.2	93.0	23.3	62.1	98.0
	SFT + RL w/ scalar f/b	4.0	31.2	62.8	10.4	56.1	94.2	28.5	64.6	98.0
	SFT + RLSF (token-level f/b)	**13.6**	**45.1**	**89.1**	**21.3**	**59.7**	**98.3**	**32.1**	**65.8**	**99.1**
`galactica-1.3b` (Taylor et al., 2022)	Zero-shot	0.0	0.0	0.0	0.0	44.8	2.2	0.0	43.2	0.2
	SFT	0.5	10.1	23.1	8.0	56.3	97.7	22.3	59.2	96.3
	SFT + RL w/ scalar f/b	2.7	27.5	75.4	11.7	56.8	**98.8**	26.1	62.7	99.0
	SFT + RLSF (token-level f/b)	**8.5**	**38.2**	**81.4**	**19.8**	**60.3**	**99.8**	**34.5**	**64.4**	**99.5**

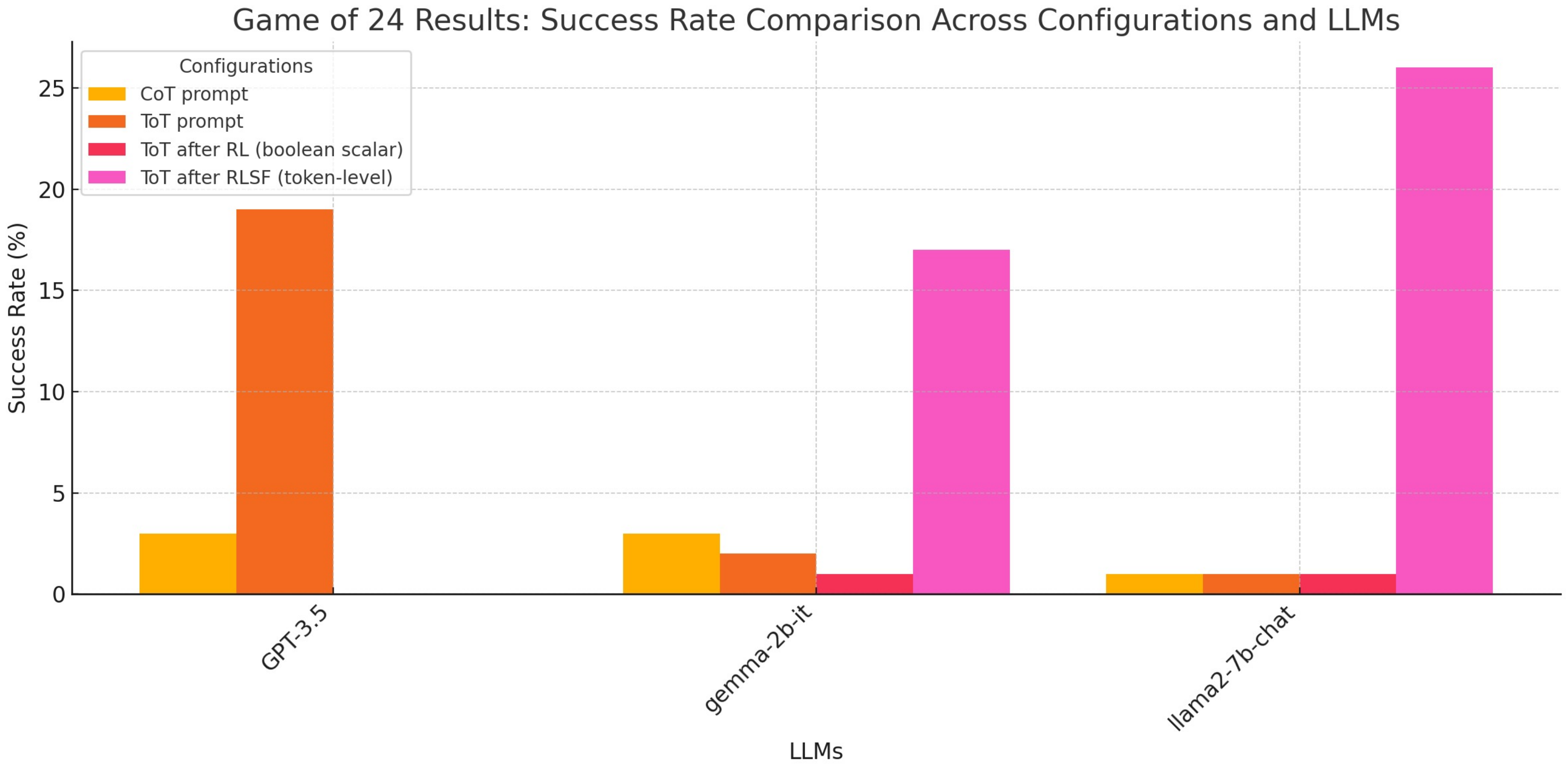


- **Molecule Generation**:
  - RLSF improved exact match by +8% and validity by +58% over SFT using **Meta's Galactica-1.3b**, also surpassing **GPT-4 (~1000× larger).**
- **Forward and Retrosynthesis**:
  - RLSF increased exact match by +19.4% for forward synthesis and +33.7% for retrosynthesis, significantly outperforming **GPT-4 (~1000× larger).**



Experimental Results: RLSF for Game of 24

CoT: Chain-of-thoughts
ToT: Tree-of-thoughts



RLSF boosted success rates by +25% on **Meta's Llama2-7b** compared to traditional methods, also surpassing **GPT-3.5 (25× larger)** with +7% improvement.

Georgia Tech

RLSF SUMMARY

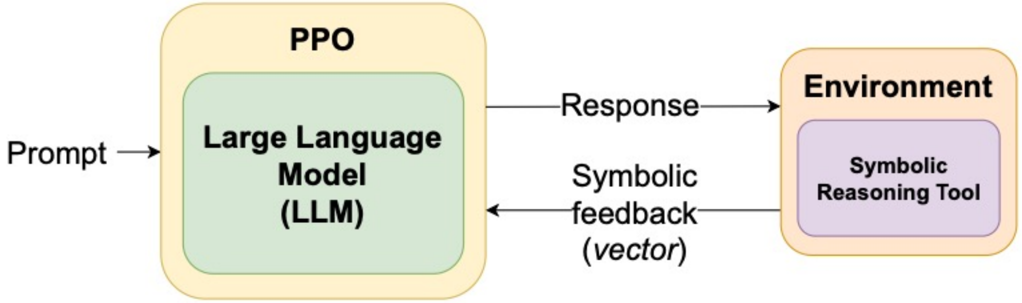
- RL Agent with Symbolic Environment
- Leverages Poly-Sized Certificates
- Fine-Grained Token-Level Feedback
- Non-Differentiable Feedback
- Improved Performance on Reasoning Tasks

Fine-grained Symbolic Feedback

Outperforms GPT4

1000x Smaller LLMs

Reinforcement Learning from Symbolic Feedback (RLSF)



Reasoning

Learning

RLSF: Reinforcement Learning via Symbolic Feedback
P. Jha, P. Jana, P. Suresh, A. Arora, V. Ganesh

Georgia Tech

CoTran: An LLM-based Code Translator using RL with Feedback from Compiler and Symbolic Execution

Prithwish Jana[a], Piyush Jha[a], Haoyang Ju[b], Gautham Kishore[c], Aryan Mahajan[d], Vijay Ganesh[a]

[a] Georgia Institute of Technology, USA
[b] University of Toronto, Canada
[c] UC San Diego, USA
[d] Columbia University, USA

ECAI
50th ANNIVERSARY
27TH European Conference on AI
19-24 October, 2024
Santiago de Compostela, Spain

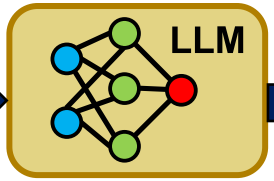
Georgia Tech
University of Toronto
UC San Diego
Columbia University

Problem Statement: LLM for Whole-Program Translation

```java
import java.io.*;
public class Main
{
    static int unitDigitXRaisedY(int x, int y)
    {
        int res = 1;
        for (int i = 0; i < y; i++) res = (res * x) % 10;
        return res;
    }
    public static void main(String args[])
    {
        System.out.println(unitDigitXRaisedY(4, 2));
    }
}
```

A **Java** (*S*) program

f_{ST}: $S \rightarrow T$



Translation of a code from one high-level language to another

```python
def unitDigitXRaisedY(x, y):
    res = 1
    for i in range(y):
        res = (res * x) % 10
    return res
if __name__ == '__main__':
    print (unitDigitXRaisedY(4 ,2))
```

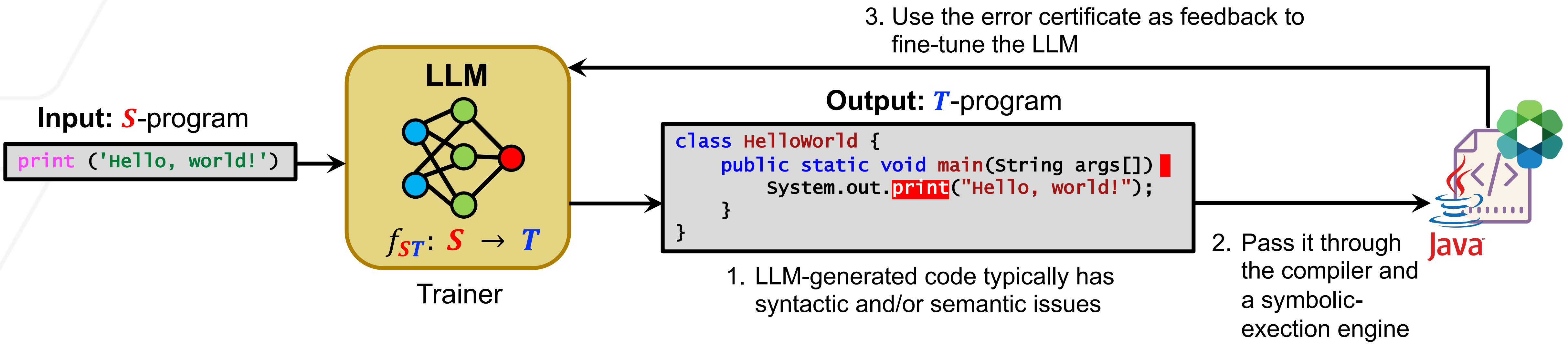
An "*equivalent*" program in **Python** (*T*)

- Let S = **source language** (e.g., Java) and T = **target language** (e.g., Python)
- **Code Translation Learning Problem**
  - Learn f_{ST}: $S \rightarrow T$, which when provided with a S-program produces a T-program that is
    - **syntactically correct** (as per the grammar of T) and
    - **functionally equivalent** (w.r.t. a test suite) to the input S-program

PROPOSED METHODOLOGY

AN OVERVIEW OF THE PROPOSED IDEA

- Improve LLM by providing **logical feedback** during fine-tuning **(neuro-symbolic approach)**

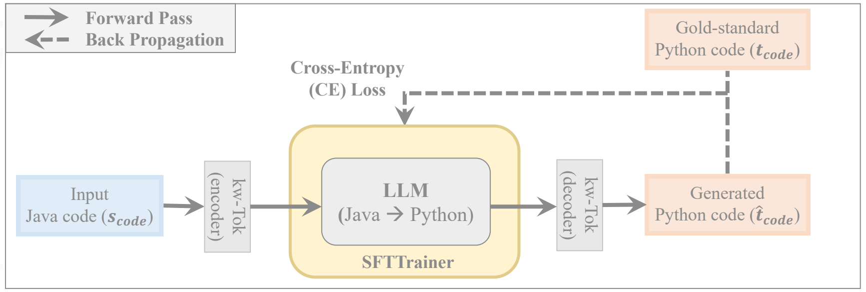


- **Compiler Feedback (CF)**
  - How close the *T*-program is to being perfectly compilable
- **Symbolic-Execution Feedback (SF)**
  - How closely the *T*-program is ‘equivalent’ to *S*-program

Scalar, but fine-grained feedback ∈ $[0,1]$

- We use **RL + Supervised Fine-tuning (SFT) interleaved training** to incorporate feedback

CoTran + Compiler Feedback (CF)



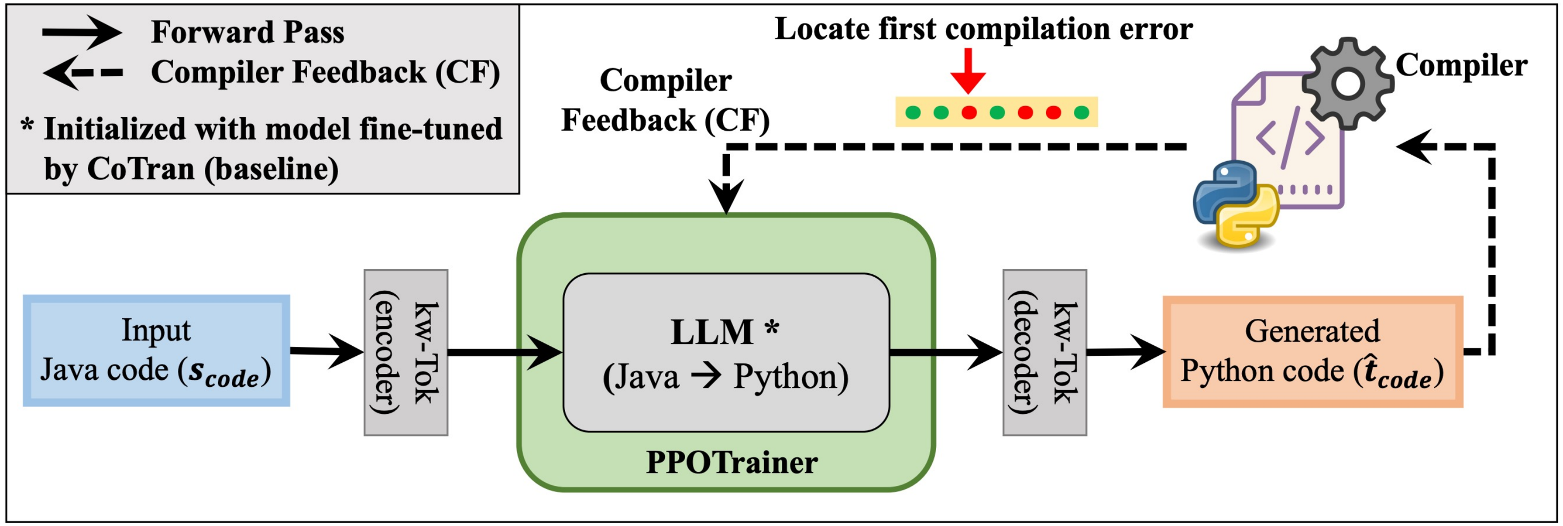
The baseline CoTran (using Supervised Fine-Tuning i.e., SFT)

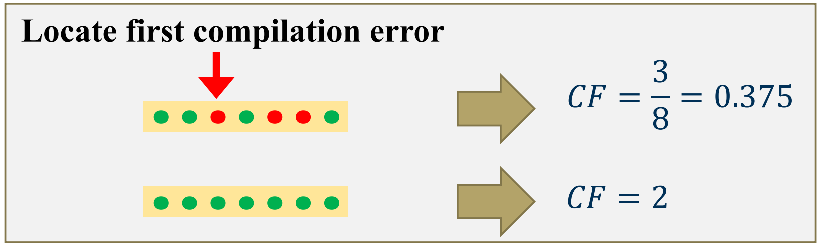
$$\mathcal{L}_{CE}^{\theta_f}\left(\mathbf{t},\hat{\mathbf{t}}\right) = -\frac{1}{\ell}\sum_{i=1}^{\ell}\sum_{j=1}^{|V|}\mathbb{1}_{ij}\log P_{ij}^{\theta_f}$$

Issue with Cross-Entropy Loss-based approach:

Cross-Entropy loss **does not penalize** based on **wrong syntax** of generated code

- RL-based fine-tuning of LLM using PPO, to **maximize Compiler Feedback (CF)**

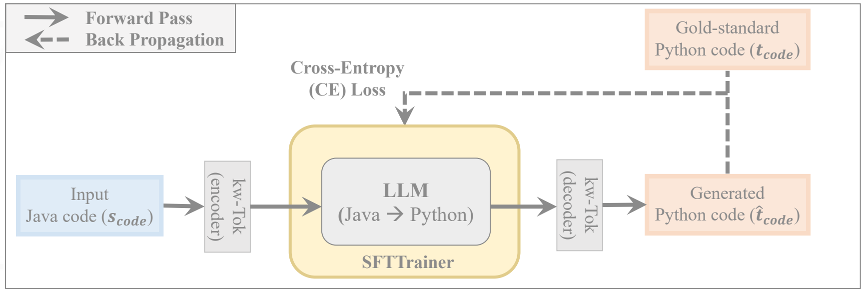




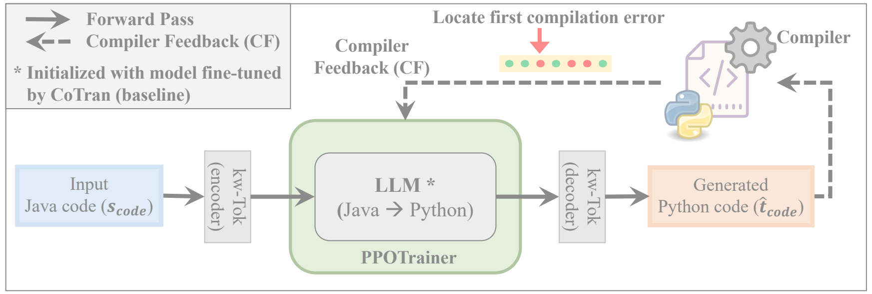
Guides the LLM into determining **how far it is** from producing a **perfectly compilable code**

- ***Cue:*** The first error token's position (the deeper it is, the closer to perfect compilation)

CoTran + CF + Symbolic-Execution Feedback (SF)



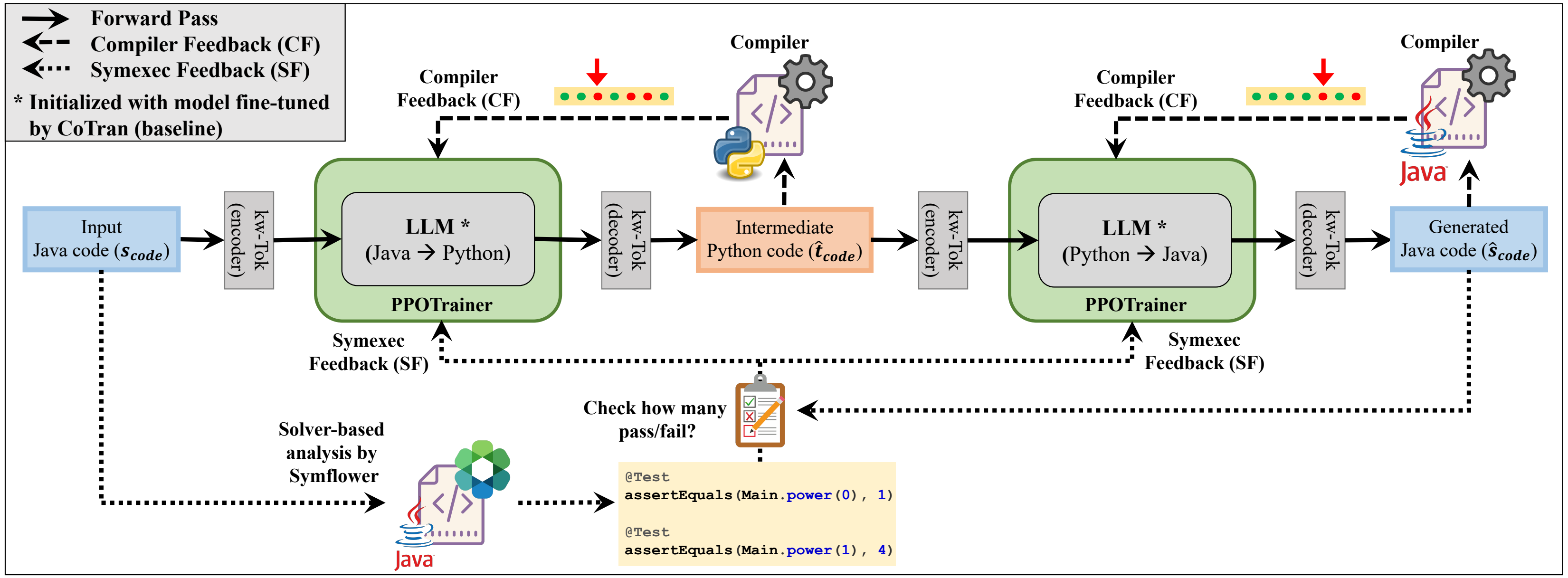
The baseline CoTran (using Supervised Fine-Tuning i.e., SFT)



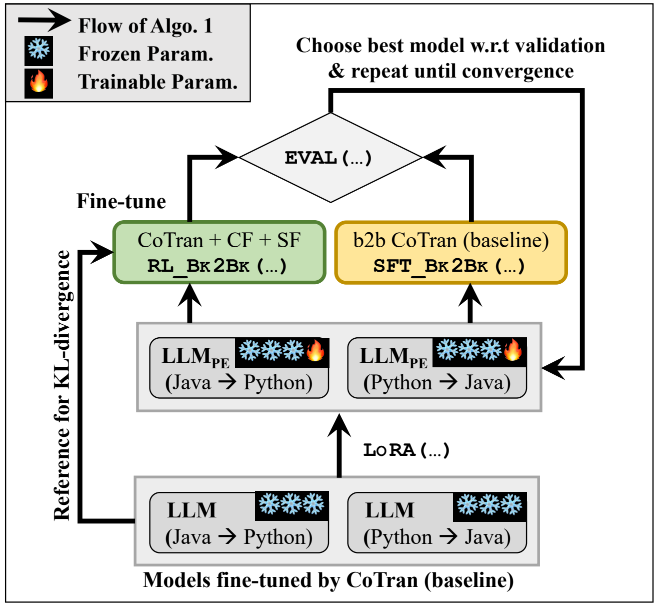
CoTran + CF (using RL-based fine-tuning by PPO algorithm)

<u>Issue with CoTran+CF:</u>

CF loss **does not penalize** based on **inequivalence** of generated and input codes



CoTran + CF + SF (using RL-based fine-tuning on back-to-back LLMs)



RL + SFT Interleaved Training Loop

ECAI 50th ANNIVERSARY

CoTran: An LLM-based Code Translator using RL with Feedback from Compiler and Symbolic Execution

P. Jana, P. Jha, H. Ju, G. Kishore, A. Mahajan, V. Ganesh

Georgia Tech
University of Toronto
UC San Diego
Columbia University

EXPERIMENTAL RESULTS

Performance Comparison for J2P and P2J Translation

Method / Tool	Model	J2P FEqAcc	J2P CompAcc	J2P $errPos_{1st}$	J2P CodeBLEU	J2P BLEU	J2P EM	P2J FEqAcc	P2J CompAcc	P2J $errPos_{1st}$	P2J CodeBLEU	P2J BLEU	P2J EM
S1 Transpilers	java2python [Melhase *et al.*, 2016]	3.32	41.46	28.62	20.31	17.54	0	-	-	-	-	-	-
	TSS CodeConv [TSS, 2023]	0.46	58.30	54.26	41.87	24.44	0	-	-	-	-	-	-
	py2java [Fomin, 2019]	-	-	-	-	-	-	0	0	1.61	41.56	48.59	0
S2 Recent competing tools (*unsupervised training*)	TransCoder [Roziere *et al.*, 2020]	0.46	88.09	63.57	35.07	32.07	0	0	0	4.57	35.02	35.06	0
	TransCoder-DOBF [Lachaux *et al.*, 2021]	0.46	63.00	47.10	39.98	33.84	0	0	0	3.11	33.33	32.72	0
	TransCoder-ST [Roziere *et al.*, 2022]	0.46	91.58	74.68	40.04	37.30	0	0	0	4.67	29.88	28.15	0
S3 ChatGPT	GPT-3.5-turbo [OpenAI, 2023]	**76.06**	95.36	90.88	52.11	53.19	0.29	21.65	24.97	30.86	54.08	55.58	0
S4 Recent competing tools (*supervised training on AVATAR-TC*)	CodeBERT [Feng *et al.*, 2020]	12.31	84.77	79.57	46.00	48.10	0.46	0.74	**96.79**	**99.51**	26.10	19.62	0
	GraphCodeBERT [Guo *et al.*, 2021]	10.88	85.05	79.78	45.53	47.26	0.57	0.46	89.75	98.05	23.72	16.21	0
	CodeGPT [Lu *et al.*, 2021]	24.86	78.92	89.21	38.38	38.64	1.49	13.40	45.13	94.50	40.51	37.96	0.52
	CodeGPT-adapted [Lu *et al.*, 2021]	24.17	76.75	89.31	36.84	37.36	1.55	20.50	52.00	97.60	41.46	38.15	1.03
	PLBART-base [Ahmad *et al.*, 2021a]	38.55	91.47	90.79	54.77	59.34	1.32	38.26	75.77	96.64	55.96	59.24	0.97
	CodeT5-base [Wang *et al.*, 2021]	40.95	92.84	**93.76**	55.34	60.03	2.41	33.79	68.84	98.02	57.64	60.16	0.86
	PPOCoder [Shojaee *et al.*, 2023]	44.27	93.47	91.44	55.16	59.51	1.89	37.11	59.62	96.77	55.04	58.52	0.52
P Our tool	**CoTran (baseline)**	44.52	96.12	92.07	55.44	58.71	2.11	40.41	73.63	92.16	**59.11**	61.12	**1.66**
Our tool with compiler feedback only	CoTran + CF (RL-based training)	47.02	96.56	91.58	56.10	60.59	2.23	42.78	74.80	96.91	58.55	61.26	1.60
	CoTran + CF (RL+SFT interleaved training)	49.83	96.79	92.08	56.07	60.61	2.23	45.93	75.77	96.89	58.28	61.21	1.60
Our tool (b2b) with compiler & symexec feedback	CoTran + CF + SF (RL-based training)	50.45	96.79	92.15	56.17	60.60	2.23	43.92	75.14	96.93	58.59	**61.28**	1.60
	CoTran + CF + SF (RL+SFT interleaved training)	53.89	**97.14**	92.73	**56.24**	**60.69**	**2.29**	**48.68**	76.98	96.93	58.38	61.19	1.60

Abbreviations (all in [0,100], higher better)

FEqAcc	:	Functional Equivalence Accuracy
CompAcc	:	Compilation Accuracy
$errPos_{1st}$	:	Average First Error Position
EM	:	Exact Match

We compare our proposed tool (P) CoTran with the following SoTAs:

- **S1**. Rule-based transpilers
- **S2**. Recent unsupervised LLM-based approaches
- **S3**. ChatGPT (`gpt-3.5-turbo`)
- **S4**. Recent supervised LLM-based approaches

J2P and P2J Translation Performance: Findings

	Method / Tool	Model	Java to Python (J2P) FEqAcc	CompAcc	$errPos_{1st}$	CodeBLEU	BLEU	EM	Python to Java (P2J) FEqAcc	CompAcc	$errPos_{1st}$	CodeBLEU	BLEU	EM
S3	ChatGPT	GPT-3.5-turbo [OpenAI, 2023]	**76.06**	95.36	90.88	52.11	53.19	0.29	21.65	24.97	30.86	54.08	55.58	0
S4	Recent competing tools (*supervised training on AVATAR-TC*)	CodeBERT [Feng *et al.*, 2020]	12.31	84.77	79.57	46.00	48.10	0.46	0.74	**96.79**	**99.51**	26.10	19.62	0
		GraphCodeBERT [Guo *et al.*, 2021]	10.88	85.05	79.78	45.53	47.26	0.57	0.46	89.75	98.05	23.72	16.21	0
		CodeGPT [Lu *et al.*, 2021]	24.86	78.92	89.21	38.38	38.64	1.49	13.40	45.13	94.50	40.51	37.96	0.52
		CodeGPT-adapted [Lu *et al.*, 2021]	24.17	76.75	89.31	36.84	37.36	1.55	20.50	52.00	97.60	41.46	38.15	1.03
		PLBART-base [Ahmad *et al.*, 2021a]	38.55	91.47	90.79	54.77	59.34	1.32	38.26	75.77	96.64	55.96	59.24	0.97
		CodeT5-base [Wang *et al.*, 2021]	40.95	92.84	**93.76**	55.34	60.03	2.41	33.79	68.84	98.02	57.64	60.16	0.86
		PPOCoder [Shojaee *et al.*, 2023]	44.27	93.47	91.44	55.16	59.51	1.89	37.11	59.62	96.77	55.04	58.52	0.52
P	Our tool	**CoTran (baseline)**	44.52	96.12	92.07	55.44	58.71	2.11	40.41	73.63	92.16	**59.11**	61.12	**1.66**
	Our tool with compiler feedback only	CoTran + CF (RL-based training)	47.02	96.56	91.58	56.10	60.59	2.23	42.78	74.80	96.91	58.55	61.26	1.60
		CoTran + CF (RL+SFT interleaved training)	49.83	96.79	92.08	56.07	60.61	2.23	45.93	75.77	96.89	58.28	61.21	1.60
	Our tool (b2b) with compiler & symexec feedback	CoTran + CF + SF (RL-based training)	50.45	96.79	92.15	56.17	60.60	2.23	43.92	75.14	96.93	58.59	**61.28**	1.60
		CoTran + CF + SF (RL+SFT interleaved training)	53.89	**97.14**	92.73	**56.24**	**60.69**	**2.29**	**48.68**	76.98	96.93	58.38	61.19	1.60

CoTran (P) improves upon CodeT5 (S4)

- Compared to CodeT5-base, in **J2P** and **P2J** respectively, CoTran + CF + SF gets **+12.94%** in J2P **and +14.89%** on FEqAcc

Incorporating **compiler and symexec feedback (CF, SF)** during fine-tuning **significantly improves LLM performance**

CoTran vs. state-of-the-art similar-sized LLMs (P vs. S4)

- On FEqAcc, CoTran gets **+9.62%** (vs. PPOCoder) in **J2P** and **+10.42%** (vs. PLBART- base) in **P2J**

CoTran outperforms all other SoTA tools of similar size for both **J2P**, **P2J**

CoTran vs. ChatGPT (P vs. S3)

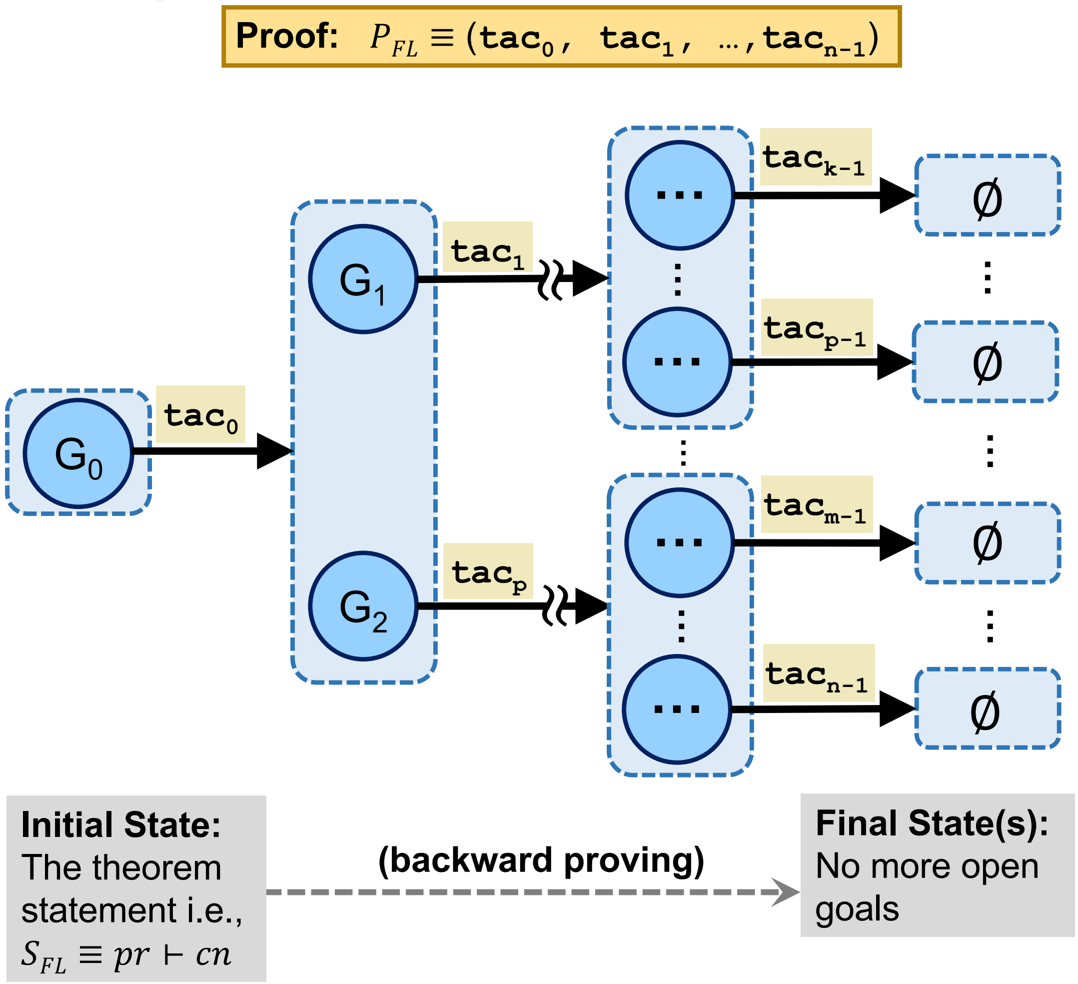
- ChatGPT, a 1000× larger model
- In **P2J**, CoTran gets +27.03% in FEqAcc, +52.01% in CompAcc

A **smaller model trained with symbolic feedback** outperforms a larger model without it.

AUTOMATED PROOF SYNTHESIS & AUTO-FORMALIZATION USING LLMS + LEAN

TACTIC-STYLE PROOFS IN LEAN4

- An imperative and procedural approach; proofs constructed backward from goal to premises
- Proof represented as a directed acyclic graph (DAG) of **proof states** and **tactics**



- Each **proof state** $S_i \equiv [G_1, \cdots, G_n]$ (dotted boxes) consists of a sequence of zero or more *open goals*.
  - Initial state S_0 has only one goal, $S_{FL} \equiv pr \vdash cn$
  - Final proof states have no open goal
- Each **open goal** $G_i \equiv pr_i \vdash cn_i$ (circles) of a proof state represents a proposition.
- Each **tactic** tac_i (directed edges) represents a proof step.
  - A high-level command (rooted in metaprogramming) applied to an open goal G_i, producing a new proof state with zero or more sub-goals.
  - If a tactic results in a proof state with no open goal, it directly resolves the current goal.
  - Parent goal gets resolved once all subgoals resolved.

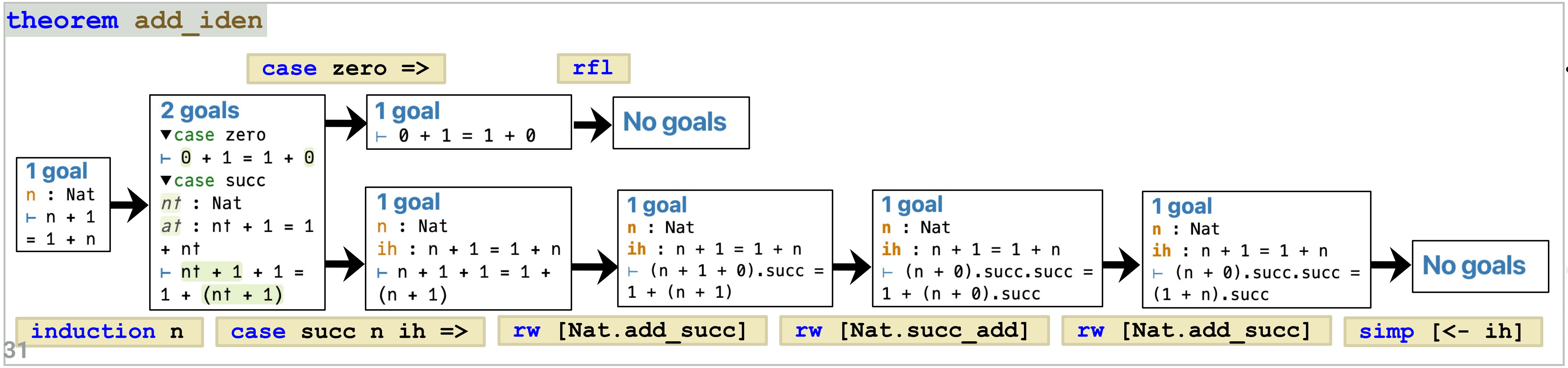
Example of Tactic-Style Proof in Lean4

```
theorem add_iden (n: Nat) : n + 1 = 1 + n := by
induction n
case zero =>
    rfl
case succ n ih =>
    rw [Nat.add_succ]
    rw [Nat.succ_add]
    rw [Nat.add_succ]
    simp [<- ih]
```

A Lean4 proof demonstrating the additive identity property for natural numbers

DAG representations of the example proof:

- Each **proof state** $S_i \equiv [G_1, \cdots, G_n]$ is a sequence of zero or more open goals
- Each **open goal** $G_i \equiv pr_i \vdash cn_i$ contains proposition cn_i to be proven, given pr_i.
- The final proof state(s) has no open goals.
- Each **tactic** (directed edge) is a proof step, a high-level command transforming an open goal to a new proof state.



LLM for Automated Proof Synthesis

- **Next Tactic (Proof-step) generation**
  - Given a current state, LLM predicts the subsequent tactic
  - **Pro**: more reliable, can verify tactic using the formal verifier to obtain updated information about the current tactic state, often utilizing tree search techniques to construct valid proofs
  - **Con**: relatively slower than whole-proof generation

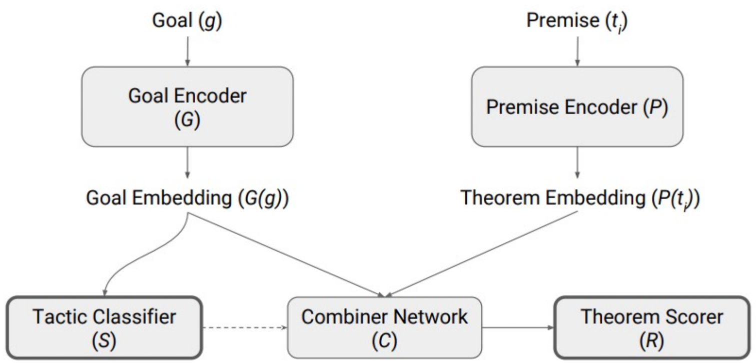
- **Whole-proof generation**
  - Given a theorem statement, LLM produces an entire proof code
  - **Pro**: computationally efficient, requires less communication budget to coordinate between the LLM and the LEAN verifier
  - **Con**: sequential nature introduces the risk of compounding errors i.e. a single wrong step can lead to significant deviations from a valid proof path.

Early AI/ML-Only Approaches to Proof Synthesis

DeepHOL (2019)

HOL Light

- 2-tower neural network to encode proof goals and premises resp.
- Goal embedding → classify tactics
- Goal-premise embedding pair → score and get useful premises (tactic arguments)
- Search via BFS



GPT-f (2020)

MetaMath

- **Proofstep objective:** LLM asked to generate the PROOFSTEP given a GOAL

`GOAL <GOAL> PROOFSTEP <PROOFSTEP><EOT>`

Input — Output

- **Proof search:** Sample 32 tactics, apply valid and deduplicated ones, prioritize by cumulative log probability → **best-first search**

L-Isa (2021)

Isabelle

- **Same as GPT-f**, specific for Isabelle

PACT (2022)

Lean

- **Similar to GPT-f**, specific for Lean

`GOAL <TacticState> PROOFSTEP <Tactic>`

Input — Output

- **Small tactic space**
  - 41 tactics
- **Pre-LLM technique**
  - Can't *'generate'* arbitrary formulas as tactic parameters

- **All these LLM-based tools use Best-first Search**
  - Relies solely on the confidence of fine-tuned LLM to predict the next proof step (tactic)
  - No guarantee that following the LLM's suggestions will lead to a successful or faster proof
- **Recent methods use neuro-symbolic AI**
  - Formal symbolic reasoning (Interactive Theorem Prover) with the LLM in a loop

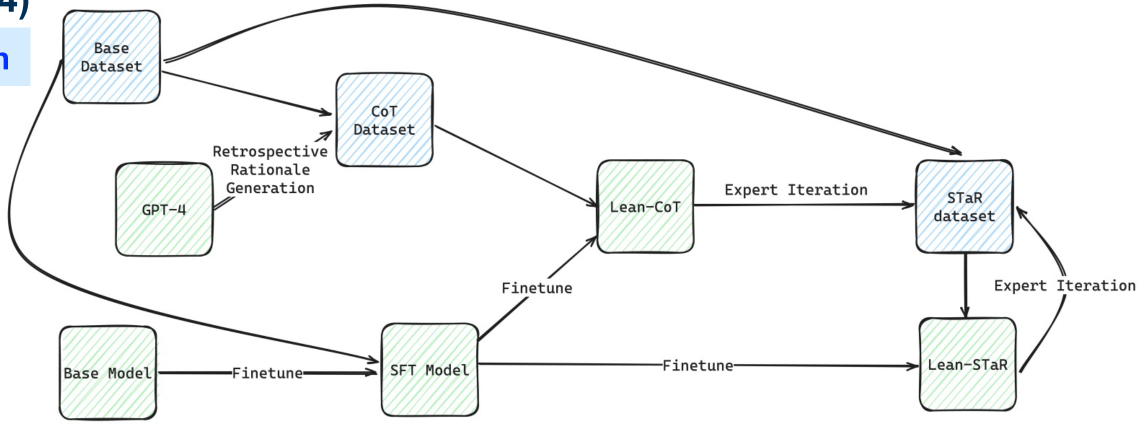
[1] Bansal, K., Loos, S., Rabe, M., Szegedy, C., & Wilcox, S. (2019). HoList: An Environment for Machine Learning of Higher Order Logic Theorem Proving. In *ICML* (pp. 454-463). PMLR.
[2] Polu, S., & Sutskever, I. (2020). Generative Language Modeling for Automated Theorem Proving. *arXiv preprint arXiv:2009.03393*.
[3] Jiang, A. Q., Li, W., Han, J. M., & Wu, Y. (2021). LISA: Language models of ISAbelle proofs. In *6th Conference on AITP* (pp. 378-392).
[4] Han, J. M., Rute, J., Wu, Y., Ayers, E. W., & Polu, S. (2021). Proof Artifact Co-training for Theorem Proving with Language Models. *arXiv preprint arXiv:2102.06203*.

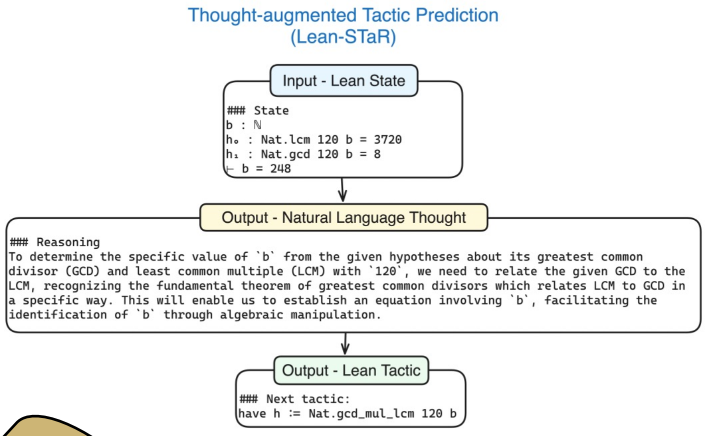
Georgia Tech

Neuro-Symbolic AI Systems for Proof Synthesis

Lean-STaR (2024)

Lean



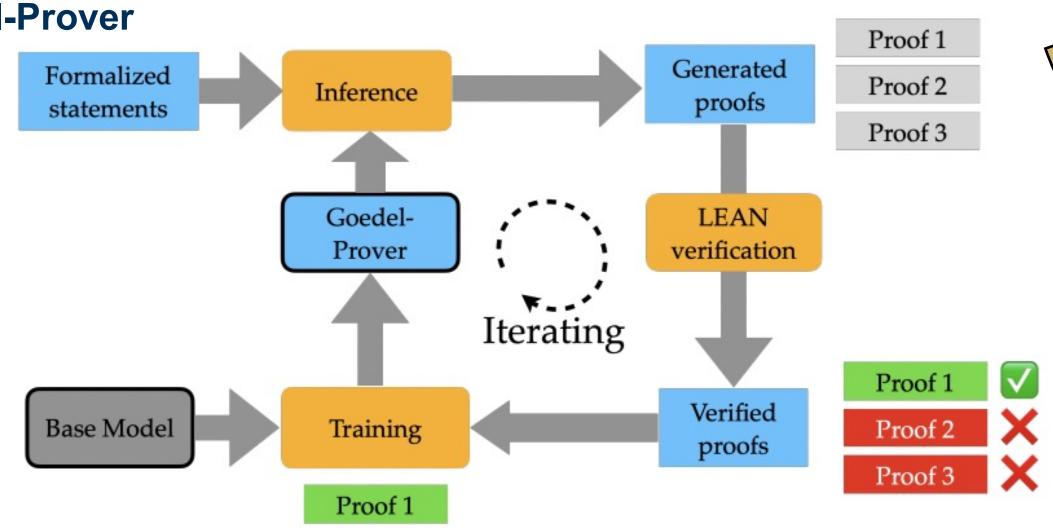


Thought-augmented Tactic Prediction

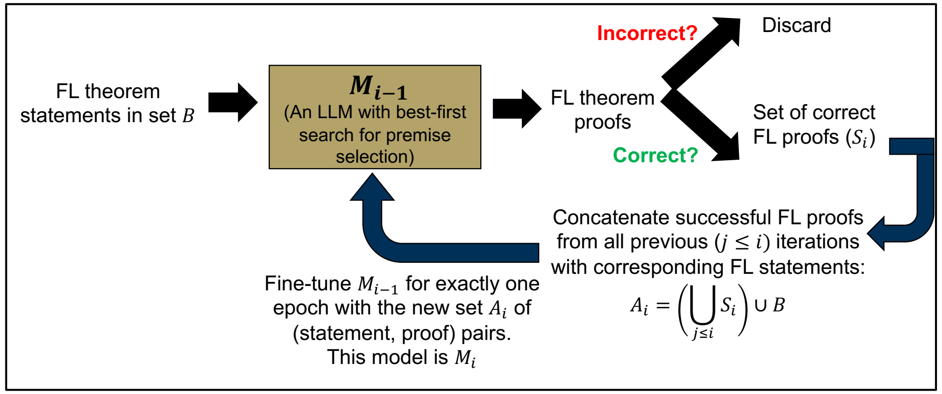
- Given a state, LLM that generates tactic along with NL thoughts (reasoning), perform better

Goedel-Prover (2025)

Lean



Both employ Expert Iteration Loop with Lean ITP:



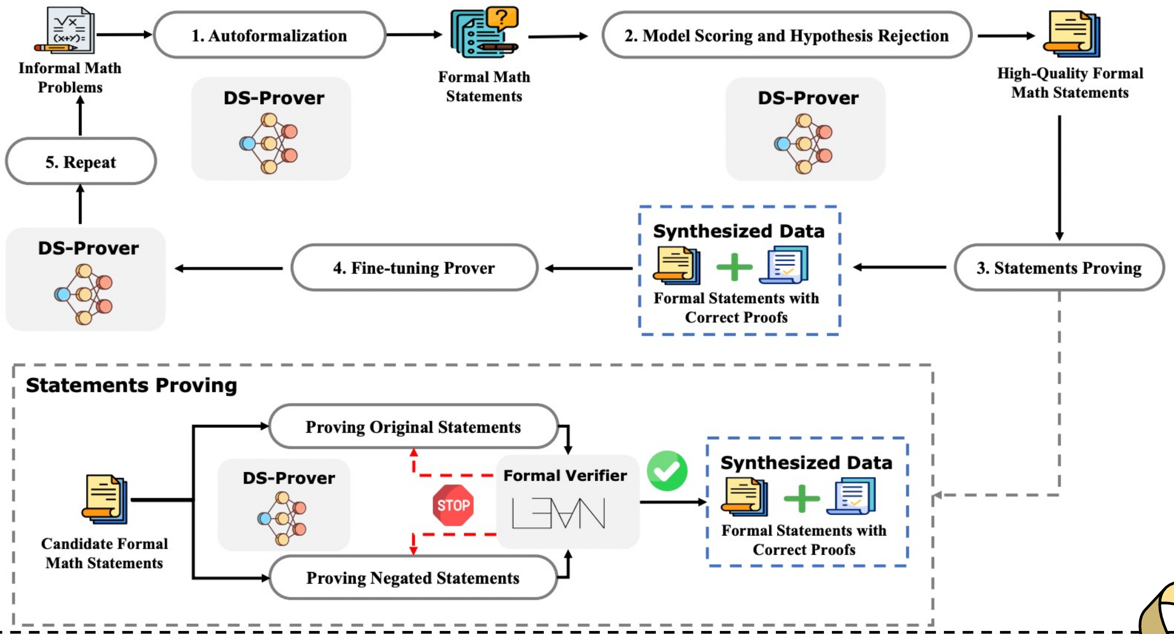
[5] Lin, H., Sun, Z., Welleck, S., & Yang, Y. (2024). Lean-star: Learning to interleave thinking and proving. *arXiv preprint arXiv:2407.10040*.

[6] Lin, Y., Tang, S., Lyu, B., Wu, J., Lin, H., Yang, K., ... & Jin, C. (2025). Goedel-Prover: A Frontier Model for Open-Source Automated Theorem Proving. *arXiv preprint arXiv:2502.07640*.

NEURO-SYMBOLIC AI SYSTEMS FOR PROOF SYNTHESIS (CONTD…)

DeepSeekProver-V1 (2024)

Lean

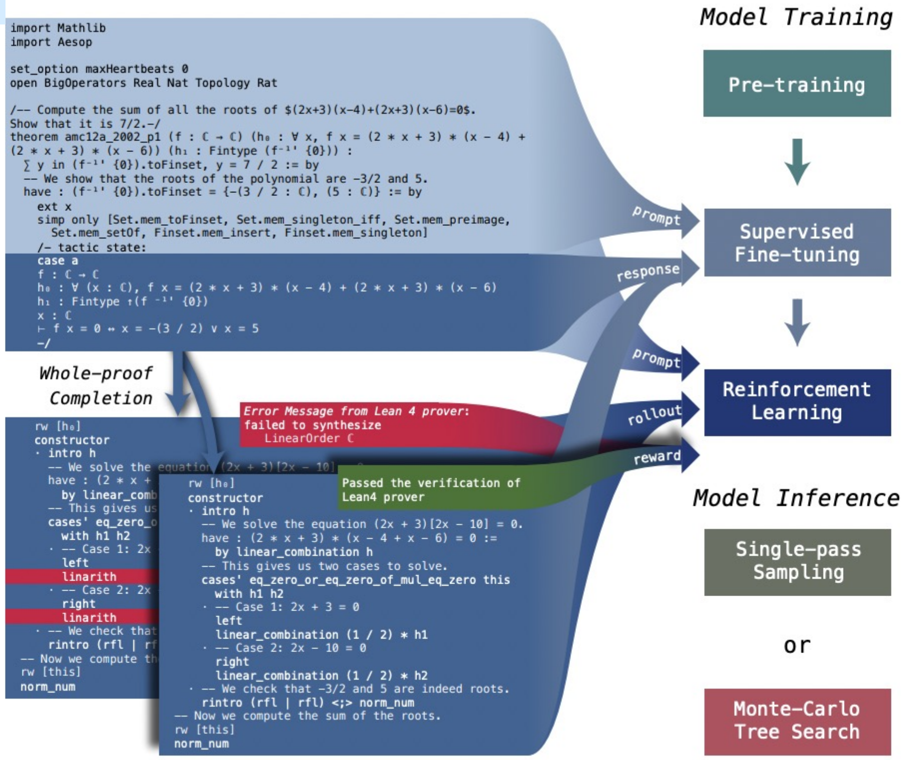


Expert Iteration Loop with Lean ITP

Reinforcement Learning training loop with Lean ITP

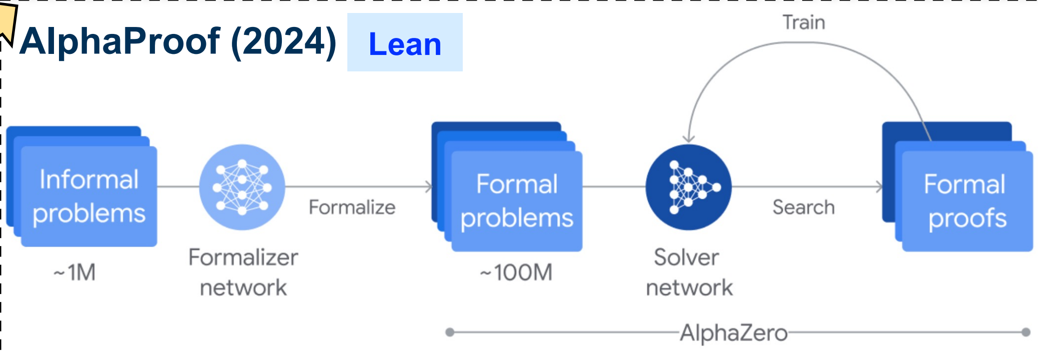
DeepSeekProver-V1.5 (2024)

Lean



AlphaProof (2024)

Lean



[7] Xin, H., Guo, D., Shao, Z., Ren, Z., Zhu, Q., Liu, B., ... & Liang, X. (2024). DeepSeek-Prover: Advancing Theorem Proving in LLMs through Large-Scale Synthetic Data. *arXiv preprint arXiv:2405.14333.*

[8] Xin, H., Ren, Z. Z., Song, J., Shao, Z., Zhao, W., Wang, H., ... & Ruan, C. (2024). DeepSeek-Prover-v1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search. *arXiv preprint arXiv:2408.08152.*

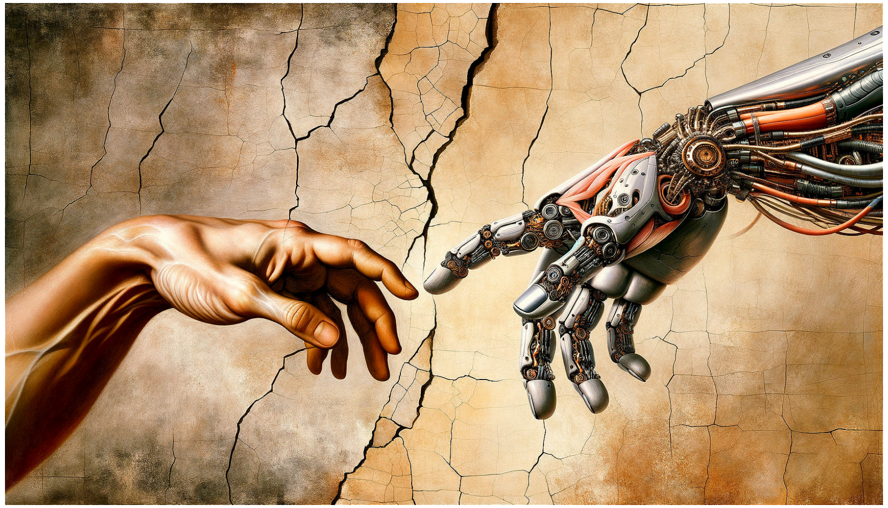
[9] AlphaProof and AlphaGeometry teams (2024). AI achieves silver-medal standard solving International Mathematical Olympiad problems. URL: https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/

Concluding Remarks

Conclusion & Future Directions

- Integrate **dense fine-grained feedback from formal verification tools** via **RLSF** during **LLM training/fine-tuning/in-context** for both **normal + reasoning tokens**
  - **Software engineering** (e.g., code synthesis, code translation and code repair)
  - **Mathematical reasoning** (e.g., proof synthesis and auto-formalization)
  - To develop specialized **neuro-symbolic** small language models **(SLMs)** → perform better than LLMs

- **Future directions:** Using LLMs in different formal math settings
  - Solving International Mathematics Olympiad problems
  - Populate mathlib library for currently un-formalized fields like cryptography and proof complexity
  - *and more hard problems…*



THANK YOU!

Georgia Tech