PieLoT — LLM-driven Toolbox for Theorem Proving Education

Ding Feng [*1] **Li Daoxin** [*1] **Zhang Qixiang** [*1] **Martin Henz** [1]

[*]Equal contribution [1]*National University of Singapore, Singapore*. Correspondence to: Martin Henz henz@comp.nus.edu.sg.

1. Introduction

In an era defined by the growth of critical digital infrastructure, the reliability of software is non-negotiable. While traditional testing is essential for debugging, it is insufficient for certifying the absolute correctness required by high-stakes systems [1]. Theorem proving offers a rigorous alternative, allowing engineers to verify that software behavior aligns perfectly with specifications via machine-checked proofs [2]. Consequently, the demand for robust proof assistants—and the engineers capable of wielding them—has surged [3, 4].

However, the democratization of formal verification faces a significant bottleneck: the steep learning curve. Mastering theorem proving demands a high degree of maturity in functional programming and advanced type theory [5], often discouraging novices. To address this pedagogical gap, we propose that the structured, goal-oriented process of theorem proving is an ideal candidate for enhancement via Large Language Models (LLMs).

We introduce *PieLoT* , an online platform engineered to accelerate the training of proof engineers. *PieLoT* alleviates the cognitive load of learning formal methods by integrating an educational IDE with intelligent assistance. Specifically, we contribute:

- An accessible web-based IDE for the Pie theorem prover;
- A dynamic visualization tool that elucidates complex proof states;
- A context-aware, LLM-driven hint system that guides learners through proof construction, fostering independent problem-solving.

2. Substantial section

2.1 The Pie Theorem Prover

Pie is an educational theorem prover designed with a minimalist philosophy [6]. It utilizes a Scheme-style syntax (S-expressions), which, while sometimes regarded as ergonomically inefficient due to its heavy use of parentheses, serves a critical pedagogical purpose. Pie's minimalism ensures there are no "black boxes"; learners can achieve a total comprehension of the logical machinery without hidden abstractions obscuring the proof process.

However, the existing ecosystem for Pie presents significant barriers to entry. The default tooling provides only a naive, raw text-based editor with minimal feedback or hint generation. Furthermore, the setup process is cumbersome, requiring students to configure a local Racket language environment. This creates substantial administrative overhead that detracts from the learning. To address these friction points while preserving Pie's transparency, we developed a zero-configuration, web-based, LLM-driven IDE that makes the language instantly accessible.

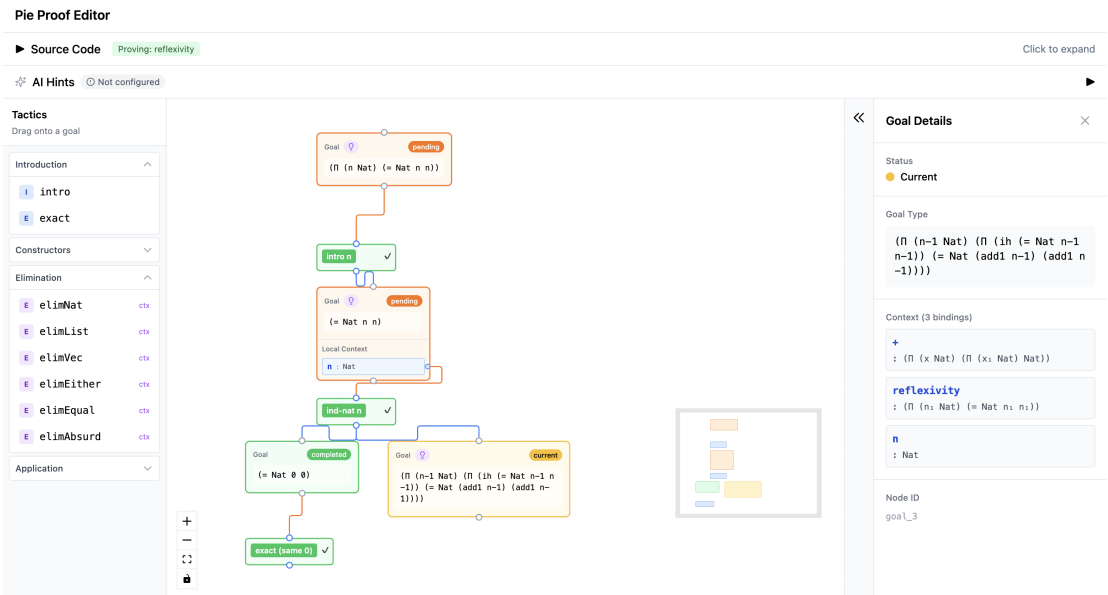

Fig. 1: The *PieLoT* Web IDE Interface. The proof tree showing a proof of reflexivity. Here, one branch of the proof is highlighted in green (completed), while the other remains in red (pending), providing immediate structural feedback to the learner.

2.2 The PieLoT Framework

We introduce *PieLoT* (Pie with LLM-driven Toolbox), a comprehensive toolkit designed to modernize the pedagogical experience of the Pie theorem prover. The framework addresses the three primary friction points in learning dependent type theory: administrative overhead, cognitive overload from syntax, and the lack of interactive feedback. *PieLoT* consists of three tightly coupled components: a zero-configuration Web IDE, a Proof Tree Visualizer, and an LLM-Driven Hint System.

2.3 Zero-Configuration Web IDE

The foundation of *PieLoT* is a browser-based Integrated Development Environment (IDE) that removes the administrative barrier of entry. Traditional Pie development requires a local Racket installation and complex environment configuration, a process that frequently discourages novices before they write their first proof.

PieLoT eliminates this step by compiling the Pie language kernel to Typescript. This allows the theorem prover to run entirely on the client side, ensuring low-latency feedback without relying on heavy server-side computation for basic type checking. The IDE features a modern code editor with syntax highlighting, automatic indentation, and immediate error reporting, mimicking the ergonomics of industry-standard tools while preserving the minimalist transparency of the original Pie language.

2.4 Interactive Proof Visualization

A core pedagogical challenge in Pie is its reliance on S-expressions (Scheme-style syntax). While syntactically minimal, this textual representation forces learners to mentally parse and reconstruct the hierarchical structure of a proof. As proofs grow in complexity, the linear text obscures the branching logic of subgoals, leading to cognitive overload.

To mitigate this, *PieLoT* includes a dynamic Proof Tree Visualizer. As the user writes a proof, the system parses the proof state in real-time and renders it as an interactive node-link diagram.

- Goal Nodes: Represent the current type that needs to be inhabited.
- Tactic Edges: Show the application of inference rules.
- Hole Status: Visually distinguishes between completed subgoals (green) and open goals (red), giving learners an immediate high-level overview of their progress.

This visualization serves as a bridge between the linear syntax of the code and the tree-structured nature of constructive logic, allowing students to "see" the proof unfold.

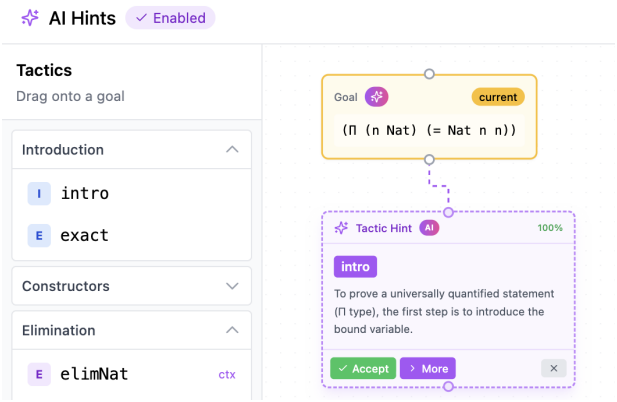

Fig. 2: Users can configure preferred LLM API within the settings and require hint when stuck on proving a goal.

2.5 LLM-Driven Hint System

The most significant innovation of *PieLoT* is its integration of a Large Language Model (LLM) as an active reasoning partner. While the visualization tool helps users understand the state, it does not tell them what to do next. Novices often suffer from “blank page syndrome”, where they understand the goal but lack the strategic knowledge to select the correct eliminator or introduction rule.

Unlike generic code assistants that simply autocomplete text, *PieLoT* ’s hint system is context-aware and goal-oriented:

Context Extraction: When a user requests a hint for a specific "hole" in the proof tree, *PieLoT* extracts the local context—including the current goal type, available assumptions in the local scope, and the proof history up to that point.

Prompt Engineering: This structured context is serialized and fed into the LLM with a system prompt specifically tuned for pedagogy. The prompt instructs the model to prioritize explanation over solution.

Strategic Guidance: The system generates a hint that explains why a certain tactic fits the current geometry of the proof (e.g., “Since your goal is a Π-type, try introducing a lambda abstraction”).

This design transforms the LLM from a solution generator into a Socratic tutor. By coupling the hint system directly with the visualizer, we ensure that the AI’s advice is grounded in the exact state the student is currently inspecting, effectively closing the loop between visualization and action.

References

[1] Edsger W. Dijkstra. Notes on structured programming. Technical Report EWD249, Technological University Eindhoven, 1970.

[2] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development: Coq’Art: The Calculus of Inductive Constructions*. Springer, 2004.

[3] Leonardo Moura and Sebastian Ullrich. The lean

4 theorem prover and programming language. pages 625–635, 07 2021.

[4] Edwin Brady. *Practical implementation of a dependently typed functional programming language*. PhD thesis, University of Durham, 2005.

[5] Per Martin-Löf. An intuitionistic theory of types: Predicative part. volume 80, pages 73–118. 1975.

[6] Daniel P. Friedman and David T. Christiansen. *The Little Typer*. MIT Press, 2018.