Theorem proving with LLMs

Overview of LeanDojo, ShannonProver, and Aristotle

2026-07-13

University of Waterloo

Outline

Proof assistants are important but hard to use

Proof assistants

- Allow computers to automatically verify the correctness of a written proof
- Usually have some limited ability to automatically complete proof. Requires a human prover to push it in the right direction (via *tactics*).
- E.g. Lean 4, Rocq, Isabelle, EasyCrypt

Mechanized proof assistants around for a long time (since 1967), but never became mainstream

- Very difficult to produce proof manually (new software proof = new paper)
- Testing, fuzzing, model checking are much less expensive ways to assure quality

Proof assistants are important but hard to use

For complex and critical systems (avionics, OS, file permission system in AWS, cryptographic schemes, etc.), testing is simply insufficient. Formal verification is *mandatory* for this level of quality assurance.

> “Program testing can be used to show the presence of bugs, but never to show their absence.”
>
> — Edsger W. Dijkstra

Proof assistants are important but hard to use

E.g. can you understand what this is saying?

```
private theorem foldl_acc_le (ds1 ds2 : List Nat) (w : Nat) (a b : Nat) (hAcc :
a ≤ b)
    (hLE : Forall₂ (· ≤ ·) ds1 ds2) :
    List.foldl (λ acc d => acc * w + d) a ds1 ≤
    List.foldl (λ acc d => acc * w + d) b ds2 := by
  match ds1, ds2, hLE with
  | [], [], .nil => exact hAcc
  | d1::ds1', d2::ds2', .cons hd htl =>
    simp [List.foldl]
    refine foldl_acc_le ds1' ds2' w (a * w + d1) (b * w + d2) ?_ htl
    exact Nat.add_le_add (Nat.mul_le_mul hAcc (Nat.le_refl _)) hd
```

Proof assistants are important but hard to use

Note that

- Many proof assistants are built on foundational mathematical systems (e.g. dependent type theory for Lean 4 and Rocq)
- Writing these proofs requires regularly diving down into obscure logical theories
- But typical SE work at high levels of abstractions: business logic, security, etc.
- Writing proofs essentially requires understanding odd PL/logic knowledge

$\Longrightarrow$ Most software engineers are not trained on, and can't use these tools!

How can LLMs help here?

- Good at discovering general patterns in vast amount of data
- Can solve open math problems by combining seemingly unrelated math theories
  - E.g. in **Disproof of the Erdős unit distance conjecture**, LLM solves a combinatorial geometry problem using algebraic number theory
  - *Personal opinion: likely discovered some sort of textual pattern between the two theories that otherwise takes a lot of sifting to find.*
- Tons of training data: pre-existing math proofs in Lean's `mathlib`. Growing body of software proofs in open libraries.

Ideally: SE describe desirable properties for software, LLMs tries to formalize and prove it for you.

How can LLMs help here?

> “We are increasingly exporing moving more pieces of the [Ethereum] protocol to a different security strategy: AI-assisted formal verification.”
>
> — Vitalik Buterin

- Describes move to prove Ethereum protocol using Lean, and translate to implementation languages.
- https://vitalik.eth.limo/general/2026/05/18/fv.html

How can LLMs help here?

> We're verifying the Signal protocol and its Rust implementation using the Lean Proof Assistant, in a public moonshot to show that formal verification of major applications is doable today. Think of it as the Liquid Tensor Experiment (LTE) for cryptography — where a complex math theorem was formalized and proved with the help of the mathematics community — except this time, we also have steadily improving AI on our side.
>
> — Beneficial AI Foundation

- https://www.beneficialaifoundation.org/blog/signal-shot

How can LLMs help here?

https://x.com/powdr_labs/status/2075237260971184348

> **powdr labs (@powdr_labs) on X**
> powdr autoprecompiles are now formally verified e2e.
>
> - Formal spec audited by humans (~500 LoC)
> - AI writes the optimizer and proves it obeys the spec
> - Drop-in FFI replacement for the Rust crate
>
> Result: on par with the hand-written optimizer, + it's FVed!
>
> Show more
>
> X (formerly Twitter) (108 kB)

```
abbrev Optimizer (p : ℕ) := ConstraintSystem p → ConstraintSystem p × Derivations p

/-- An optimizer is correct if, for every input constraint system, replacing it with the optimized
    system is both sound and complete, and the optimizer respects the degree bound. -/
def Optimizer.isCorrect (optimizer : Optimizer p) (busSemantics : BusSemantics p) : Prop :=
  (∀ originalCS : ConstraintSystem p,
    let (optimizedCS, derivations) := optimizer originalCS
    (optimizedCS.isSoundReplacementOf originalCS busSemantics) ∧
    (optimizedCS.isCompleteReplacementOf originalCS busSemantics derivations))
  ∧ optimizerRespectsDegreeBound busSemantics optimizer
```

LLMs $\iff$ Proof assistants complement each other's weakness!

- LLMs: hallucinates/unreliable $\longrightarrow$ check with proof assistant
- Proof assistant: expensive to prove anything substantial $\longrightarrow$ generate proofs with LLMs

Outline

Big Idea

- Most LLM-based theorem provers run fully autonomously on a backend server — no human in the loop
- But current LLMs still fail on novel/cross-domain theorems
- **Insight:** humans provide mathematical intuition; LLMs handle the tedious search for tactics and premises

Key design goal

- Run LLM inference **natively inside Lean**, packaged as a regular Lean tactic
- No separate Python server, no extra setup — feels like any other Lean dependency

Methodology

ReProver: the model inside LeanCopilot

LeanCopilot's default model is **ReProver** — a retrieval-augmented tactic generator (ByT5-small, 299M params).

Two-stage training

1. **Retriever** (ByT5 encoder + avg-pool): given a proof state s, rank candidate premises p_i by cosine similarity
2. **Generator** (ByT5 encoder-decoder): takes `[top-100 retrieved premises] ++ [proof state]` as input, generates the next tactic

Methodology

Two improvements over typical dense passage retrieval (DPR)

- **Accessible premises only**: restrict to the 33K premises actually in scope (vs. 128K total), using LeanDojo's program analysis
- **In-file hard negatives**: during training, sample some negatives from the **same file** as the ground truth
  - easy to confuse and make the retriever sharper

Methodology

Beam search and best-first search (ML recap)

Best-first search

- Maintains a **priority queue** (frontier) of partial proof states
- At each step, expands the **lowest-cost** node
  - in our case, by $-\log P(\text{tactic} \mid \text{state})$ as decided by the LLM
- Explores the globally most promising path at every step
- Complete: will find a proof if one exists (given enough time/memory)

Methodology

Beam search

- Best-first search but:
  - Prunes all but the k highest-scoring partial sequences — trades completeness for efficiency
- Operates **locally** per decoding step (not globally across the whole search tree)
- In tactic generation: each “beam” is a partial tactic token sequence; the final outputs are the k complete tactics returned to the caller

Methodology

Tools

`suggest_tactics` next-step suggestion

- Feeds current proof state into ReProver; **beam search** decodes and returns top-k tactic candidates
  - Context: each node in the beam search corresponds to a sequence of tokens for a tactic string, such as `exact Nat.add_le_add hAcc (Nat.le_refl _)`
- Each candidate is **speculatively executed** in Lean (meta-programming); error-producing ones are dropped
- Surviving tactics shown to user

Methodology

`search_proof` full proof best-first search

- **Nodes**: proof states (remaining goals)
- **Edges**: tactics weighted by $-\log P(\text{tactic} \mid \text{state})$
- `suggest_tactics` is the edge-expansion function: called at each node, returns k candidate tactics with their beam scores
- Best-first search picks the globally cheapest frontier node, expands it, repeats until no goals remain or timeout
- `aesop` uses the same search but with a **fixed** rule set; `search_proof` makes candidates and weights state-dependent via the LLM

Methodology

`select_premises` finding relevant lemmas

- Lean's mathlib has 189K+ theorems; locating the right one is the main bottleneck for humans
- Encodes proof goal, multiplies against pre-computed premise embedding matrix ($O(1)$ extra forward pass)

Evaluation

Evaluated on **Mathematics in Lean** (168 theorems, avg. 5.85 tactics/proof).

Experiment: enter ground-truth tactics one at a time; after each, try the tool on the remaining goals.

- **Autonomous rate**: % of theorems the tool proves with zero human-entered tactics
- **Steps automated**: % of proof steps handled by the tool (averaged over all theorems)

Evaluation

Tool	Autonomous rate	Steps automated
`aesop` (baseline)	—	40.1%
`suggest_tactics`	—	58.4%
`search_proof`	**63.7%**	**74.2%**

- `search_proof` needs only **2.08** manual tactics on average (vs. 3.86 for `aesop`)
- **85%** better than rule-based `aesop`; **27%** better than single-step suggestion

Outline

LeanDojo

- LeanCopilot is the **user-facing tool**
- LeanDojo is the **research infrastructure** expansion of the tool

Three contributions of LeanDojo *on top of LeanCopilot*:

1. **Data extraction** from Lean source code
2. **LeanDojo Benchmark** with a novel data split ⟵ **We'll just focus on this**
3. **Gym-like interaction API** for training and RL

How LeanDojo benchmark avoids contamination

LeanDojo Benchmark (from mathlib)

- 98K theorems, ~130K premises, ~150K tactics
- `random` split: **easy to game**
  - mathlib has clusters of near-identical theorems for slightly different properties of the same concept, often with **identical proofs**
  - random split scatters these across train/test: model memorizes one and trivially proves the other.
- `novel_premises` split: **aware of similar theorems**
  - if two theorems share any premise, they are **both** placed on the same side of the split
  - test theorems must use at least one premise never seen in training

Outline

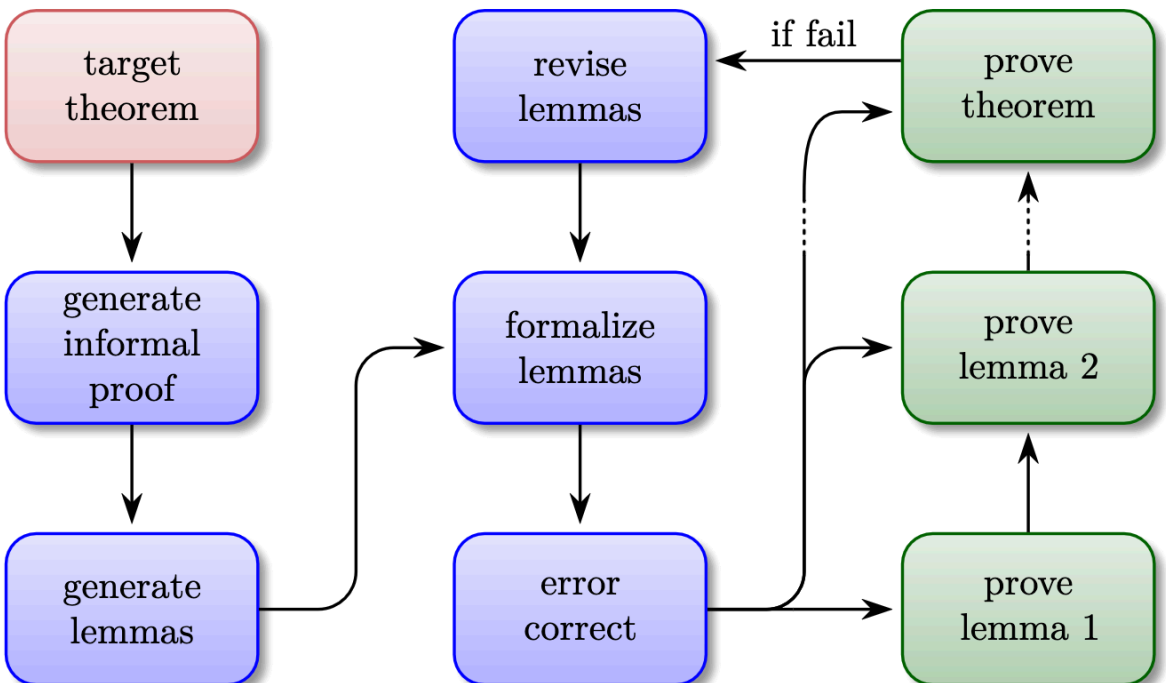
What is Aristotle?

- Automated theorem proving system built by **Harmonic**
- Achieved **gold-medal-equivalent** performance on **IMO 2025** (5/6 problems, **formally verified in Lean 4**)
  - Key distinction from other gold-medal systems (OpenAI, Google DeepMind)

Three subsystems

1. **Lean proof search**: Monte Carlo Graph Search (MCGS) guided by a large transformer (200B+ params)
2. **Lemma-based informal reasoning**: generates informal proofs, decomposes into lemmas, formalizes each, iterates with formal feedback
3. **Geometry solver**: Yuclid, a C++ DD/AR engine based on AlphaGeometry (up to 500x faster than AG-1)

What is Aristotle?



Monte Carlo Graph Search

Key engineering choices

- **Graph search** (not just tree): equivalent proof states are merged $\Longrightarrow$ avoids redundant work
- **Progressive widening**: gradually expands the action space rather than sampling all at once
- **Test-time training (TTT)**: iteratively retrains on its own search traces during inference

Lemma-based Reasoning Pipeline

Given a hard target theorem the search algorithm alone can't close:

1. Generate an **informal proof** in natural language
2. Restructure it as a sequence of **short, provable lemmas**
3. **Autoformalize** each lemma statement into Lean 4
4. Send to Lean REPL, **error-correct** based on feedback
5. Run search on each lemma; feed proved lemmas as context for subsequent ones
6. If target still fails: **revise** unproved lemmas and repeat

Lemma-based Reasoning Pipeline

Why this works

- Informal reasoning catches high-level strategy; formal feedback catches logical errors
- Novel auxiliary definitions emerge naturally (e.g. a set S of primes, a function $f(k) = k\frac{\sqrt{2}}{2k-1}$), not suggested by the problem statement

Results & Broader Impact

IMO 2025

- Solved 5/6 problems with complete, gap-free Lean 4 proofs
- Problem 6 (hardest) unsolved; also unsolved by Seed-Prover, OpenAI, and Google DeepMind

Beyond competition math

- Contributed new theorems to **Mathlib** (Niven's theorem, Gauss-Lucas theorem, eigenvalue-characteristic-polynomial correspondence)
- Found **4 false exercises** in Terence Tao's Lean real analysis textbook (provided explicit counterexamples)

- Proved statements in **homological algebra** and **Eisenstein series** (well beyond IMO-level)

Takeaway: combining LLM-guided search with formal verification is a credible path toward AI mathematical research assistants