Pythagoras Prover: Advancing Efficient Formal Proving via Augmented Lean Formalisation

Joshua Ong Jun Leang

PhD Student @ Imperial College London

j.ong25@imperial.ac.uk

Motivation: Chain-of-thought reasoning is not always faithful and reliable

Background · why reasoning needs a verifier

1 **CoT encourages multi-step reasoning**

In short, CoT instructs a model to write out its steps and improves accuracy on multi-step problems.

2 **The written steps can be unfaithful**

However, the chain often misaligns with how the model reached its answer. Here (back then in 2025), CoT **concludes 9.11 > 9.9**.

3 **Verifying the chain is therefore required**

We need a **verifier that checks each step**, not a judgement of how plausible it reads.

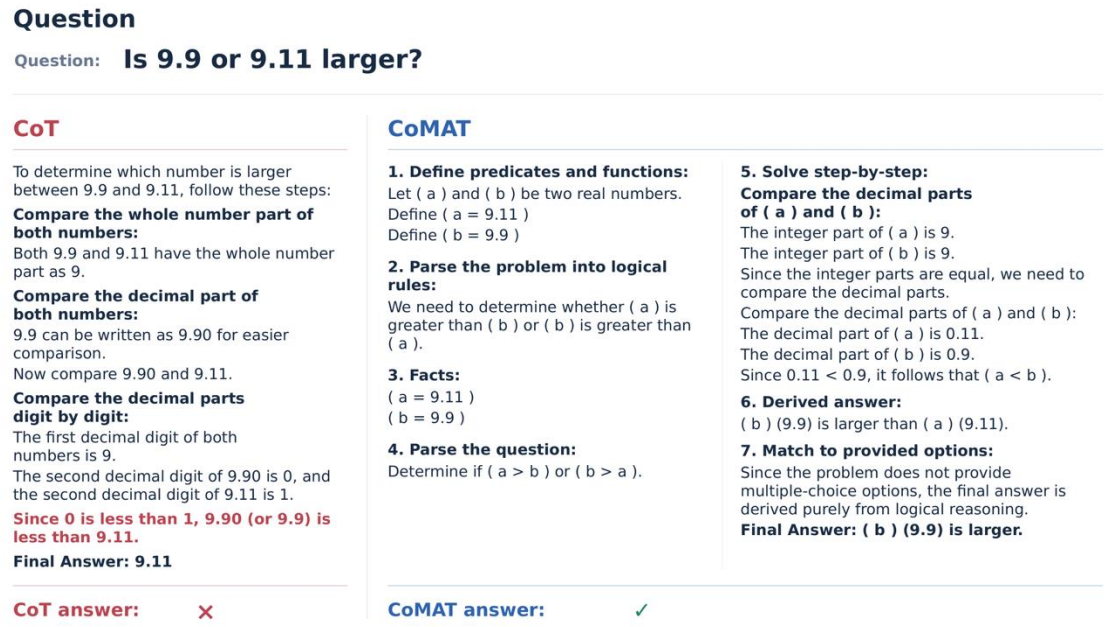


Figure: unfaithful chain-of-thought reasoning

Motivation: Theorem proving can verify a reasoning chain

Background · formal verification as the reasoning chain verifier

1 **Deterministic check**

Lean deterministically **accepts or rejects a proof.**

2 **Correct statement, correct proof**

If the formal proof is verified by the Lean compiler, we claim that the **Lean-verified proof is correct**.

3 **Applying to informal reasoning chains**

As such, formalising each step tells us **whether a chain holds**.

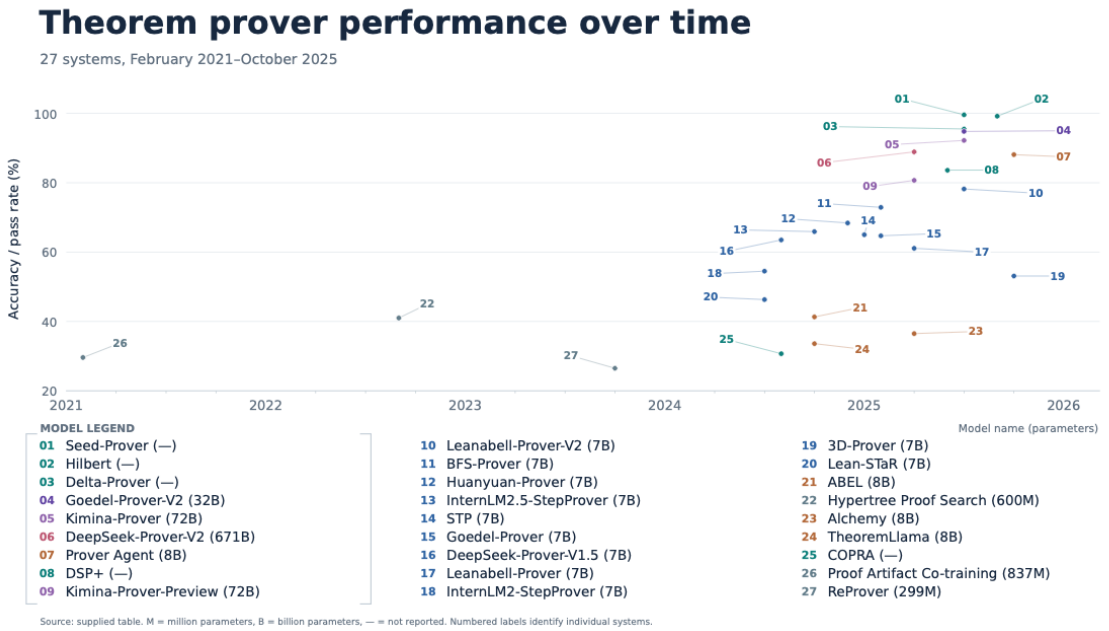


Figure: theorem prover accuracy over time, 27 systems, Feb 2021 to Oct 2025

Example of Verification in Grounding Mathematical Reasoning

Example · **the Magenta pipeline, formalising natural-language reasoning end to end**

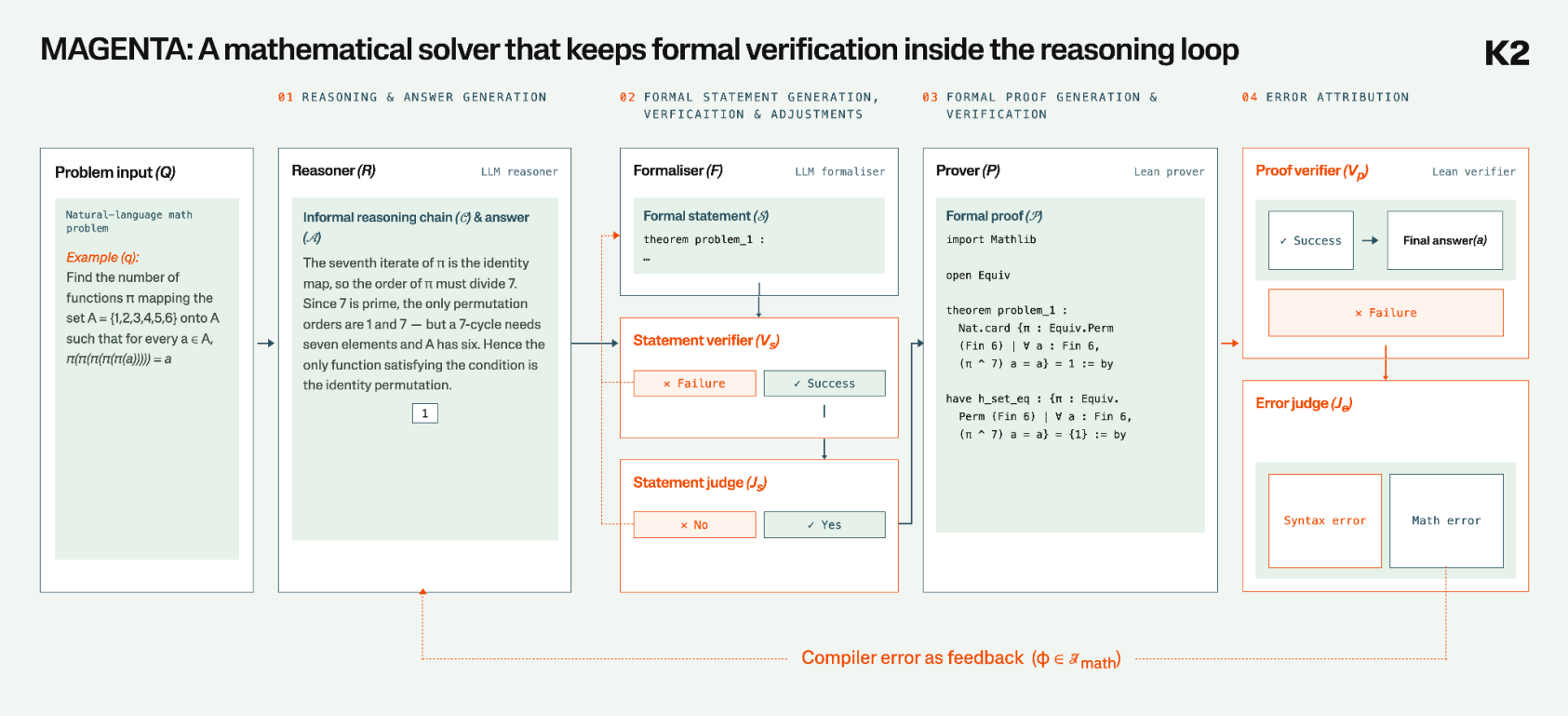


Figure: reasoning and answer generation, formal statement generation and adjudication, proof generation and verification, error attribution

Source: Magenta, https://arxiv.org/pdf/2609.11319

With the correct pipeline and feedback, all reasoner models score 100%

Results · AIME 2025, AIME 2026, HMMT Feb 2026

Perfect score on every benchmark

Four reasoner models, including a 7B model, reach **100% on all three benchmarks** with Magenta.

The smallest model gains the most

K2-Horizon-7B rises from **74.2% to 100%** overall, a 25.8-point gain.

Model	AIME 2025	AIME 2026	HMMT Feb. 2026	Overall
Claude Opus 5	100.00	100.00	96.97	98.92
Gemini 3.7 Flash	100.00	100.00	90.91	96.77
Kimi K3	93.33	90.00	78.79	87.10
K2-Horizon-7B	83.33	80.00	60.61	74.19
+ MAGENTA	**100.00** (↑ 16.67)	**100.00** (↑ 20.00)	**100.00** (↑ 39.39)	**100.00** (↑ 25.81)
K2-Horizon-375B	93.33	90.00	72.73	84.95
+ MAGENTA	**100.00** (↑ 6.67)	**100.00** (↑ 10.00)	**100.00** (↑ 27.27)	**100.00** (↑ 15.05)
Qwen3.8-27B	**100.00**	86.67	87.88	91.40
+ MAGENTA	**100.00** (↑ 0.00)	**100.00** (↑ 13.33)	**100.00** (↑ 12.12)	**100.00** (↑ 8.60)
GPT-5.6-Sol (Codex)	90.00	86.67	78.79	84.95
+ MAGENTA	**100.00** (↑ 10.00)	**100.00** (↑ 13.33)	**100.00** (↑ 21.21)	**100.00** (↑ 15.05)

Table: benchmark pass rates, %, with and without Magenta

Source: Magenta, https://arxiv.org/pdf/2609.11319

Even a 7B reasoner is capable of scoring full marks on IMO 2026

Results · IMO 2026

Perfect

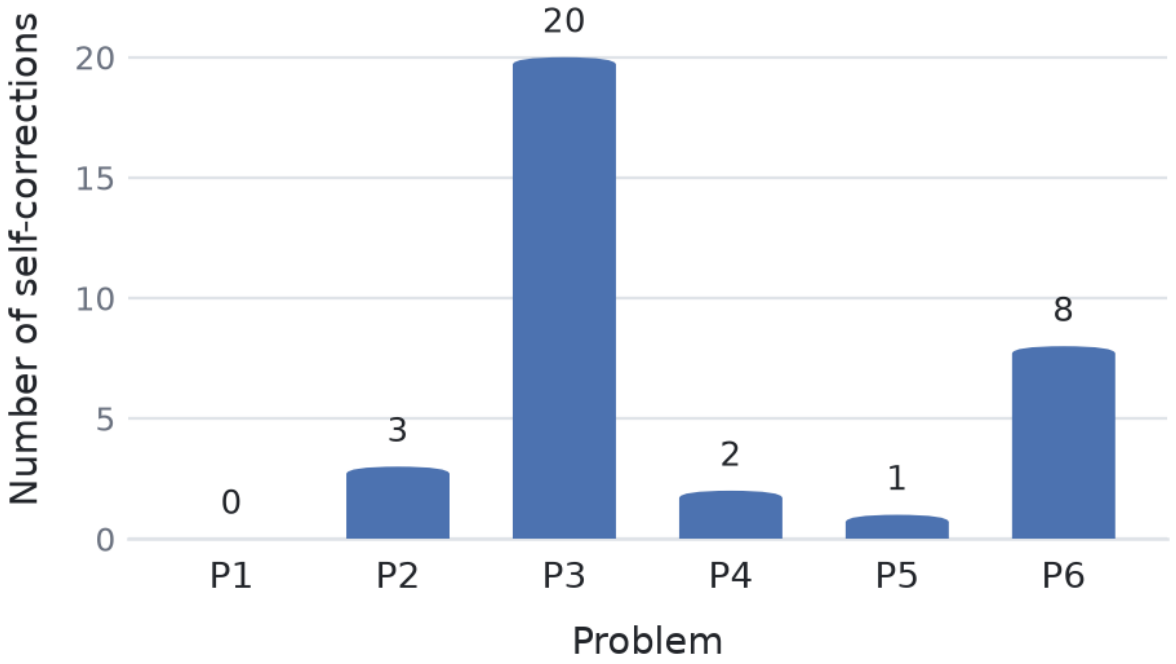
score on IMO 2026, paired with K2-Horizon-7B as the reasoner model

Smallest

model to date to reach a perfect IMO 2026 score

Lean feedback, not contamination

The model corrects itself against Lean errors: **34 corrections in total**, 20 of them on P3.



Figures: IMO 2026 results and iteration counts

Source: Magenta, https://arxiv.org/pdf/2609.11319

However, strong theorem provers are incredibly expensive to run

Limitations of current theorem-prover LLMs

1

Accuracy is no longer the gap

Foundation models for provers advanced incredibly rapidly through 2024 and 2025, and now **score highly on current benchmarks**.

2

Closed source or very large

Most SOTA provers are closed or huge, e.g. **DeepSeek-Prover-V2 at 671B** parameters.

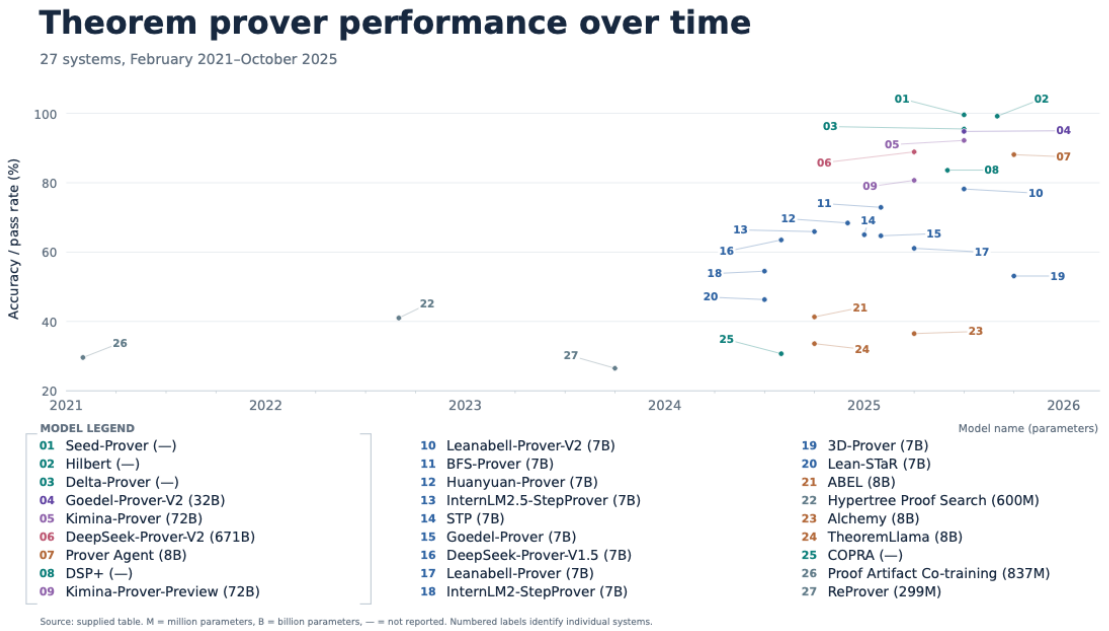


Figure: theorem prover accuracy over time, 27 systems, Feb 2021 to Oct 2025

Source: Magenta, https://arxiv.org/pdf/2609.11319

Introduction to Pythagoras Prover

A family of theorem provers with efficient training methods and inference

1 **Smallest theorem prover to date**

Pythagoras-4B outperforms DeepSeek-Prover-V2-671B on MiniF2F while being 167x smaller.

2 **First diffusion-based theorem prover**

It generates 2-3x faster than autoregressive LLMs.

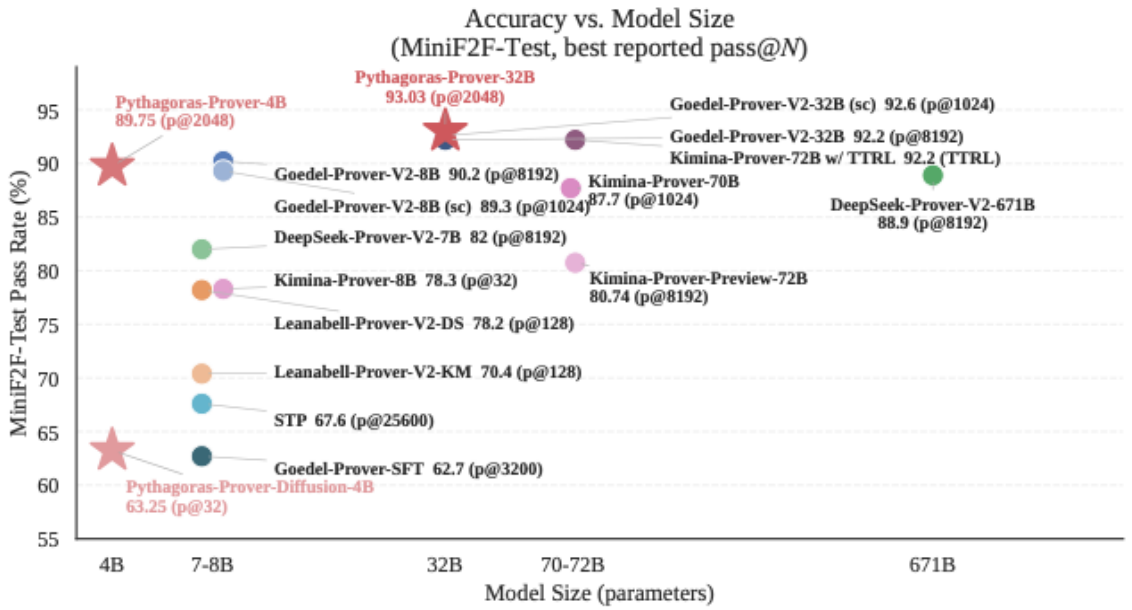


Figure: MiniF2F-Test pass rate against model size

Source: Pythagoras Prover project page, https://pythagoras-lm.github.io/

The Training Algorithm Consists of 3 Core Stages

Three core stages

1 **Synthetic Data Generation**

Synthetic data generation, combined with curriculum-learning-based training.

2 **Initial Model Training**

A novel efficient-SFT run, followed by GRPO + DAPO filtering method training.

3 **Self-Distillation with ALF**

Self-distillation with Augmented Lean Formalisation, mutating the original instances by 3x, introducing a large corpus of 3M Lean instances, open-sourced.

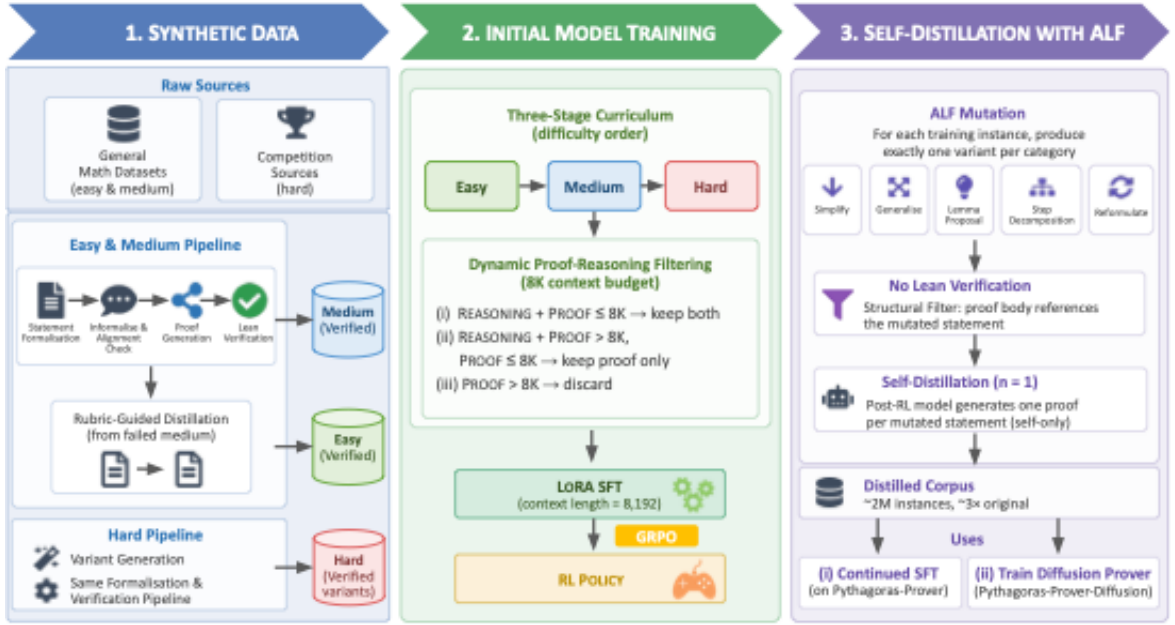


Figure: training pipeline

Source: Pythagoras Prover project page, https://pythagoras-lm.github.io/

Synthetic Data Generation

Stage 1 of 3 · Data generation pipeline

1 **Formalise the question**
Each question is first translated into a formal Lean statement.

2 **Theorem prover as a judge**
A formalise–informalise alignment check verifies the statement, then the proof is generated and Lean-verified.

3 **Recover from failures**
On failure, we extract the Lean error, analyse it against a rubric, and prompt an LLM for a simpler, targeted question.

4 **Rerun the process again**
We then rerun steps 1 and 2 on the simpler question.

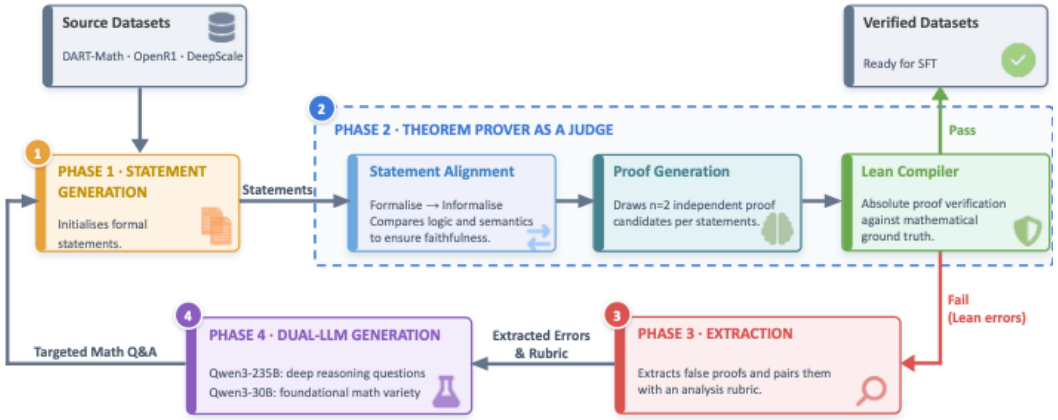


Figure: synthetic data generation pipeline

Source: Pythagoras Prover project page, https://pythagoras-lm.github.io/

Training Algorithm

Stage 2 of 3 · Training algorithm

1 **Synthetic Data Generation**

After synthesising the data, we train with a curriculum: easy instances first, then medium, then hard.

2 **Dynamic Proof-Reasoning Filtering**

We introduce an efficient training method and observe that it achieves similar performance to a model trained at 32k length.

3 **Reinforcement Learning**

We perform GRPO, coupled with the DAPO filtering method, to improve consistency and efficiency.

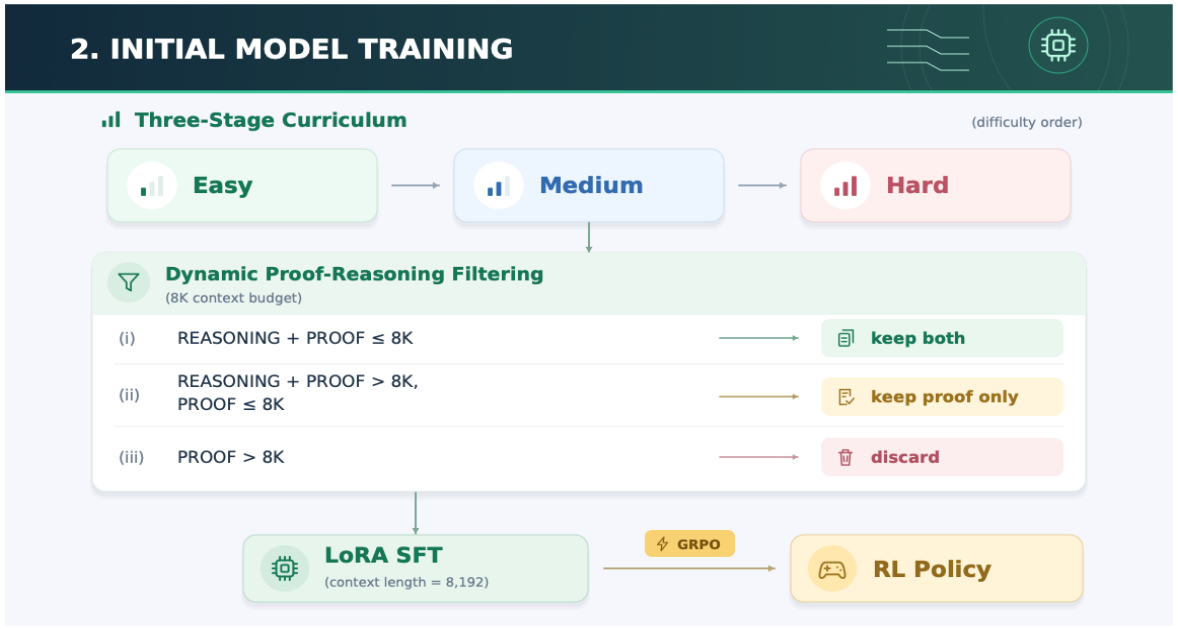


Figure: training pipeline

Source: Pythagoras Prover project page, https://pythagoras-lm.github.io/

Self-Distillation with Augmented Lean Formalisation

Stage 3 of 3 · Self-distillation

1 **What ALF is**

A mutation model (Qwen3.6-27B) rewrites each Lean statement into **new formal variants**, statements only, with no proofs yet.

2 **Augmented Lean Formalisation**

ALF applies five mutation methods to mutate the original instances by 3x, expanding the training set without new source data.

3 **Self-Distillation Training**

We train the data back into the original model, which generates the proofs from the mutated statements.

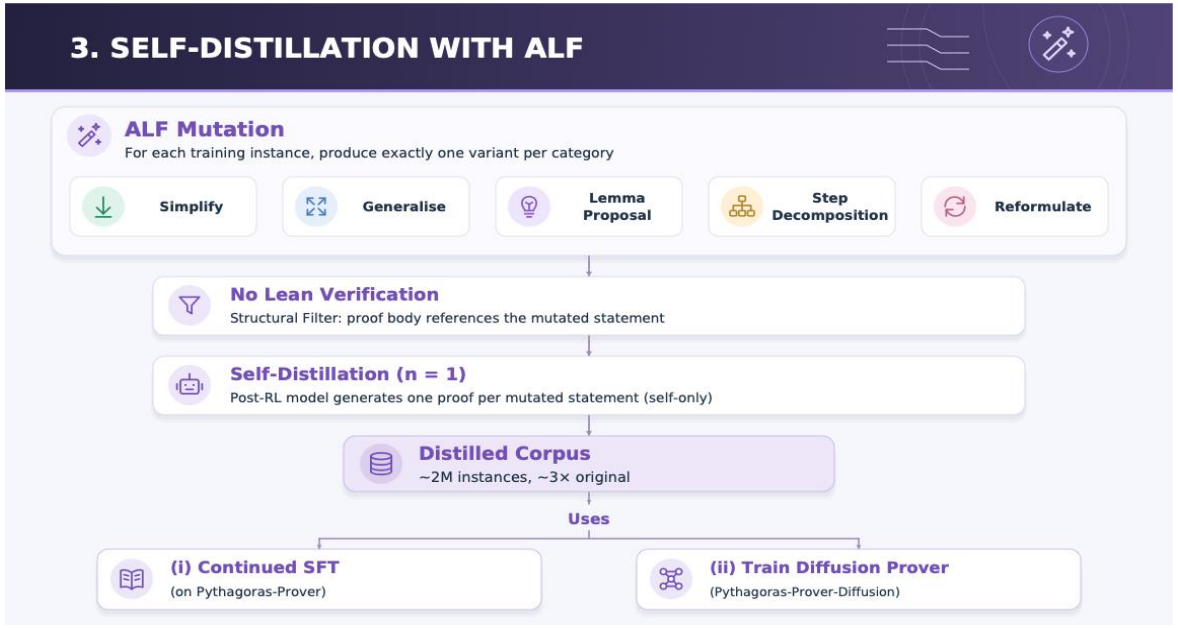


Figure: self-distillation with ALF

Source: Pythagoras Prover project page, https://pythagoras-lm.github.io/

Performance

Results · MiniF2F-ALF, PutnamBench, and accuracy against model size

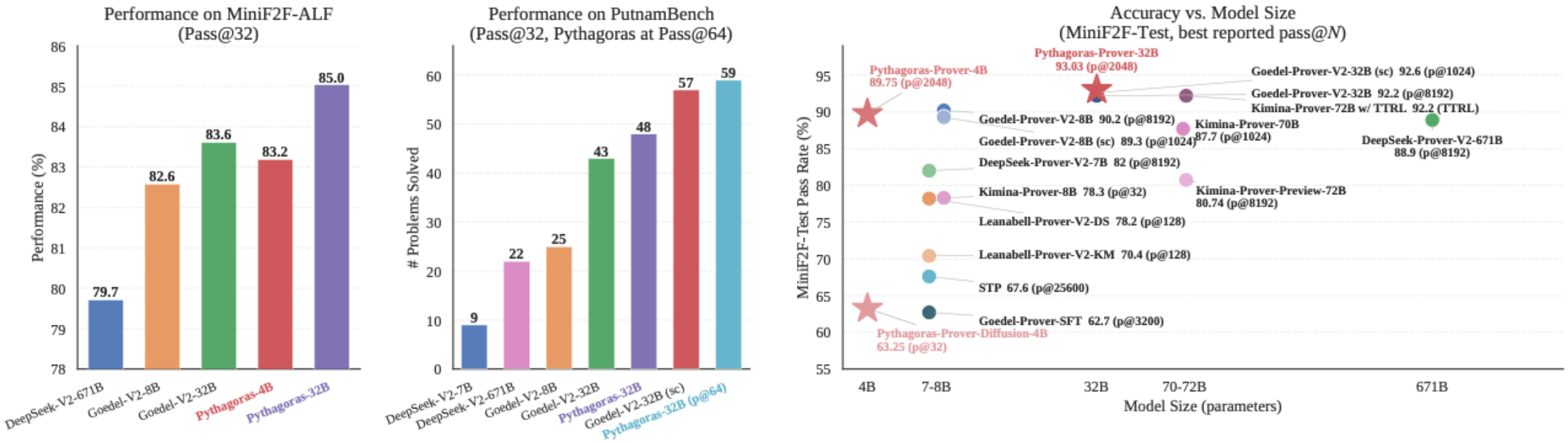
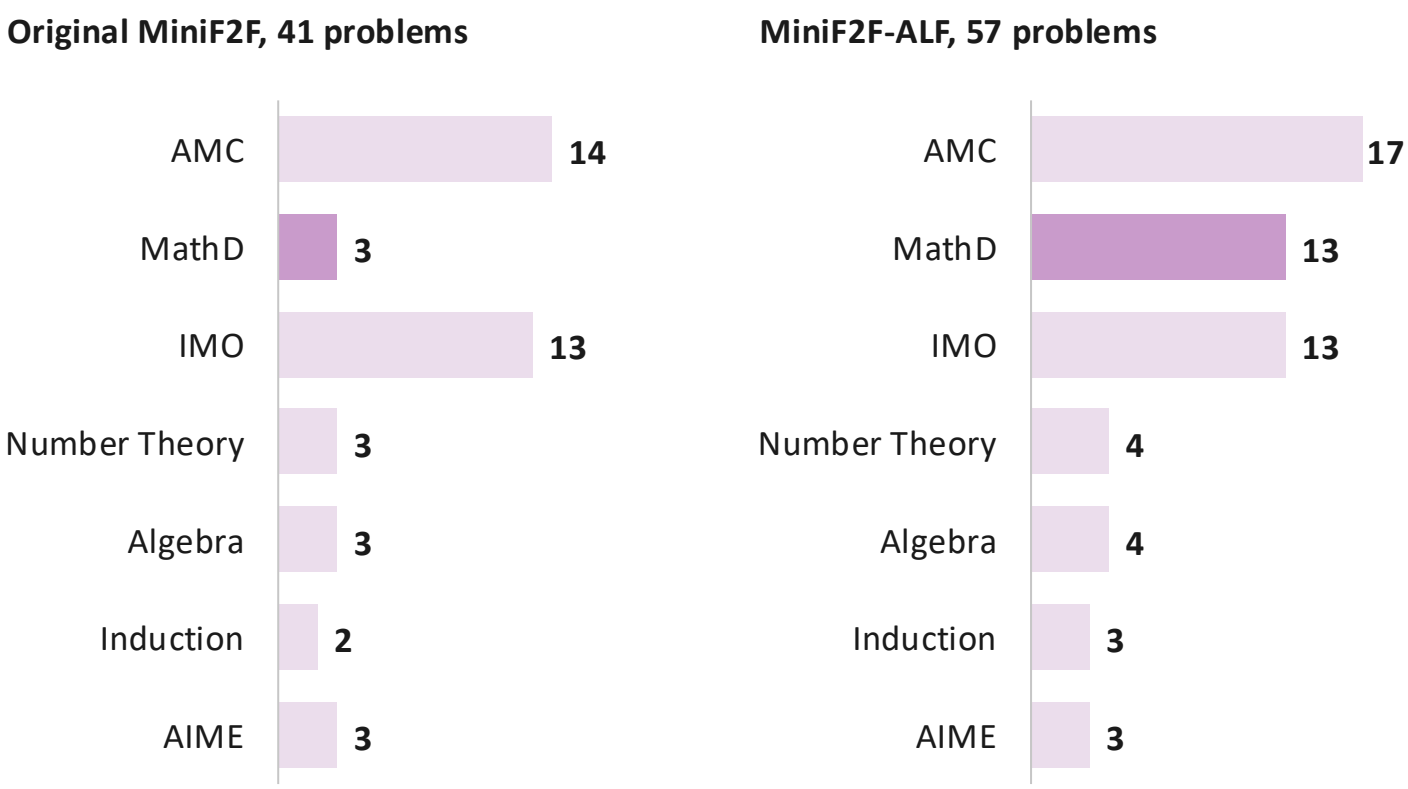


Figure: pass rates on MiniF2F-ALF and PutnamBench, and test pass rate against model size

Source: Pythagoras Prover project page, https://pythagoras-lm.github.io/

ALF mutation spreads failures beyond olympiad problems

Analysis 2 · ALF applied to MiniF2F: problems that at least one of four provers fails, by domain



Counts of problems at least one model fails; the four provers are Pythagoras-4B/32B and Goedel-Prover-V2-8B/32B

Failures rise from 41 to 57

The all-model failure rate rises from **9.0% to 13.9%** of 488 problems.

MathD failures jump 3 to 13

MathD goes from **7% to 23%** of failures, though it should not be hard under robust reasoning.

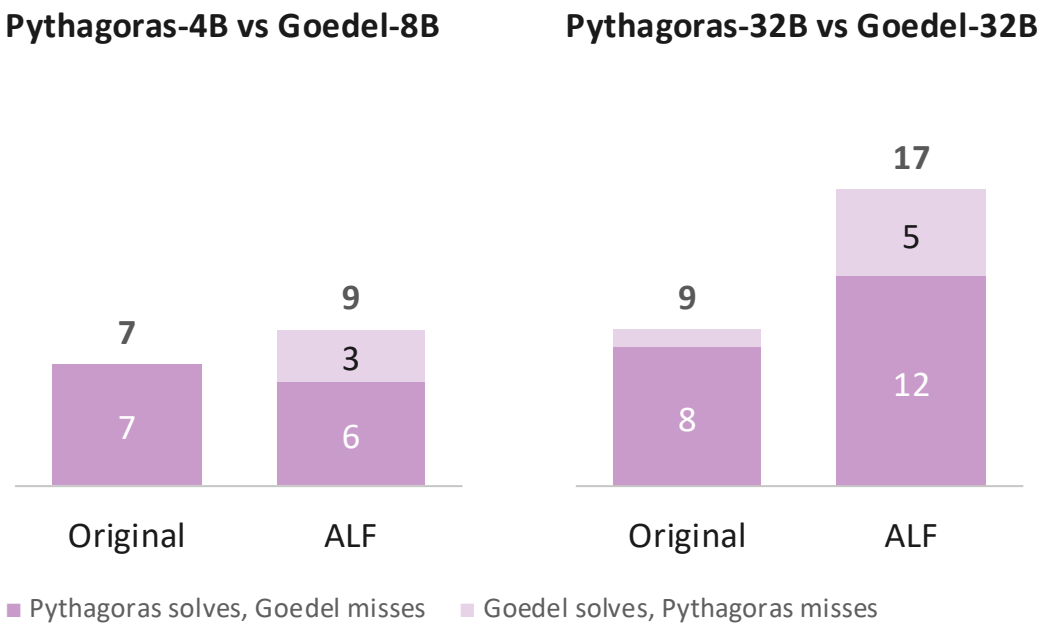
The olympiad concentration breaks

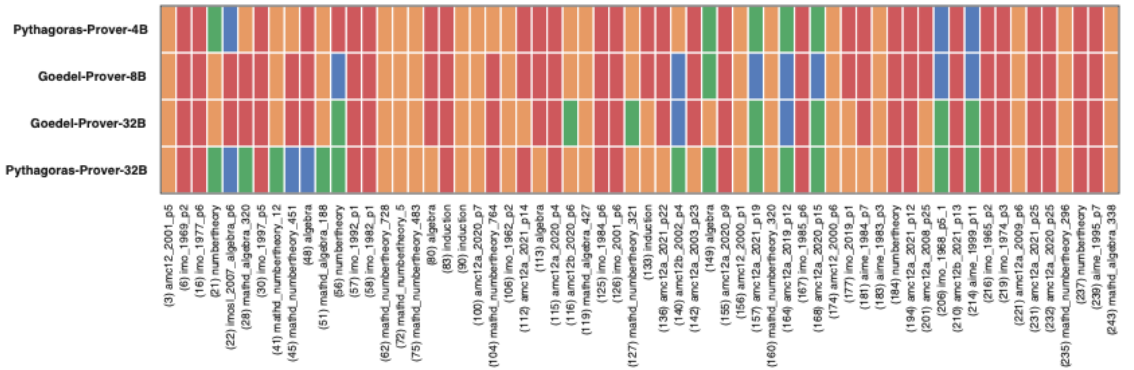
AMC and IMO fall from **66% to 53%** of failures as Algebra, Number Theory and Induction grow.

Source: Pythagoras Prover project page, https://pythagoras-lm.github.io/

Mutation makes strong provers easier to tell apart

Analysis 2 · Head-to-head disagreements, original MiniF2F vs MiniF2F-ALF





Per-problem outcome for each prover on MiniF2F-ALF; orange marks problems solved on the original but failed after mutation

Disagreements roughly double

The 32B pair disagrees on **17 problems vs 9** on the original; the 4B/8B pair on 9 vs 7.

Orange exposes hidden brittleness

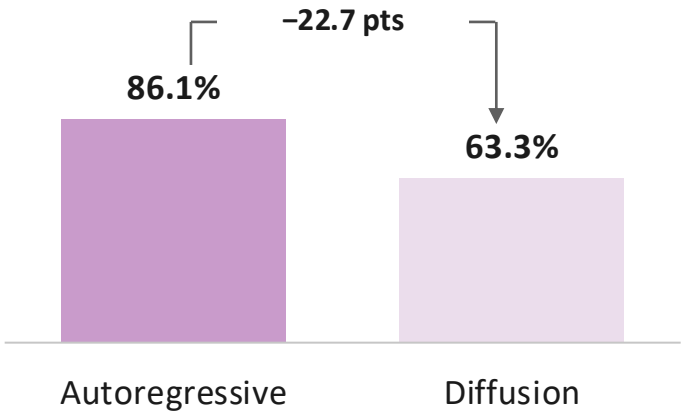
Problems solved on the original but **failed after mutation** are what high MiniF2F scores hide.

Source: Pythagoras Prover project page, https://pythagoras-lm.github.io/

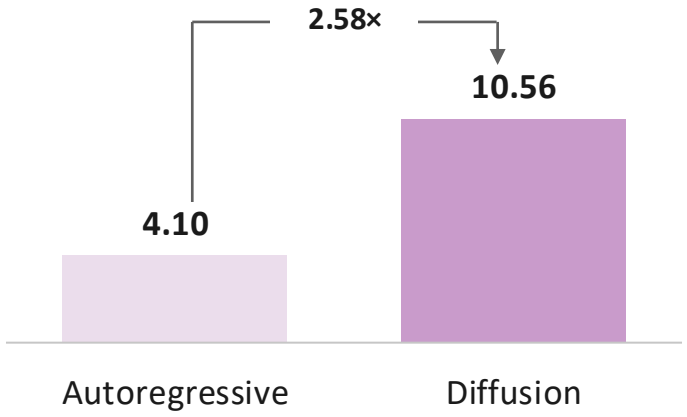
Diffusion trades accuracy for 1.9× more proofs per GPU-second

Analysis 3 · Compute efficiency, MiniF2F pass@32, autoregressive vs diffusion 4B prover

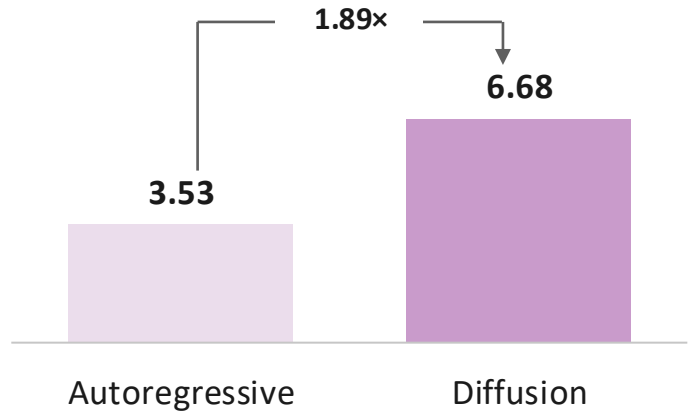
Accuracy, pass@32



Throughput, tokens/sec per GPU worker



Throughput-weighted score (accuracy × TPS)



Autoregressive when reliability matters

The **22.7-point accuracy gap** is decisive when each attempt has to succeed.

Diffusion when throughput is the constraint

Block-parallel decoding runs **2.58× faster** on the same hardware, and the margin should grow as diffusion context lengths increase.

Matched setting: one 8×H100 node, batch size 4, max_new_tokens 30,000, same self-distillation corpus; the models differ only in decoding. The score tracks proofs per GPU-second only if both generate a similar number of tokens per attempt.

Source: Pythagoras Prover project page, https://pythagoras-lm.github.io/

Summary

What this talk covered

1

Motivation

Chain-of-thought is not always faithful, so we verify it with theorem proving. Grounded in Lean, reasoners **score 100% with our Magenta pipeline.** However, strong provers are closed or very large.

2

Pythagoras Prover

Pythagoras-4B **beats DeepSeek-Prover-V2-671B** on MiniF2F at every sampling budget while being 167x smaller.

3

Three training stages

Synthetic data with curriculum learning, an efficient SFT run with GRPO, then self-distillation with ALF, which yields an **open-sourced corpus of 3M** Lean instances.

4

Analysis

MiniF2F-ALF **roughly doubles** the problems on which strong provers disagree, and the diffusion variant delivers **1.9x more proofs** per GPU-second.

Formal verification makes reasoning checkable, and Pythagoras Prover makes it affordable at 4B parameters.

Thank you

Any questions?

On a side note: I'm looking for internships in 2027, so feel free to contact me if there are any suitable positions!

References

Magenta https://arxiv.org/pdf/2609.11319

Pythagoras Prover https://pythagoras-lm.github.io/