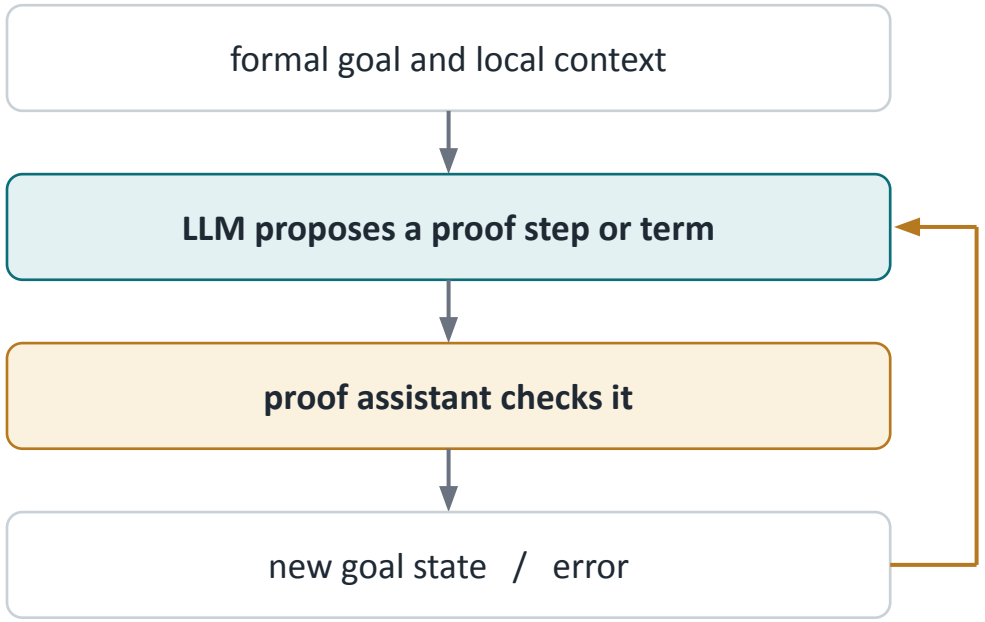
From Chatbot to Proof Agent

Search, tools, verifier feedback, and the limits of automation

Théo Stoskopf

Summer school “Proof assistants and applications” 2026

Where lecture 1 stopped



Lecture 1 ended on this interface. The question of today:

What system do we build around the model so that this interaction can continue for minutes, hours, days?

today: open this loop

The model proposes.

The harness remembers and searches.

The proof assistant verifies.

One proof is not full formalization

“Prove this isolated theorem.”

- self-contained statement;
- success = one kernel-accepted proof;
- the shape of most benchmarks.

“Formalize this 20-page argument.”

- dozens of definitions and auxiliary lemmas;
- a dependency graph, not one goal;
- must integrate into an existing library;

The assistant baseline, 2024 – early 2025

What the widely deployed assistant could do

- broad knowledge;
- follows instructions;
- writes plausible local pieces of code;
- sometimes solves hard isolated problems.

What it could not do reliably

- keep an objective over a long task;
- remember what already failed;
- maintain a consistent state across many edits;
- decide when it is actually done.

A knowledgeable assistant without consistency.

Five questions for today

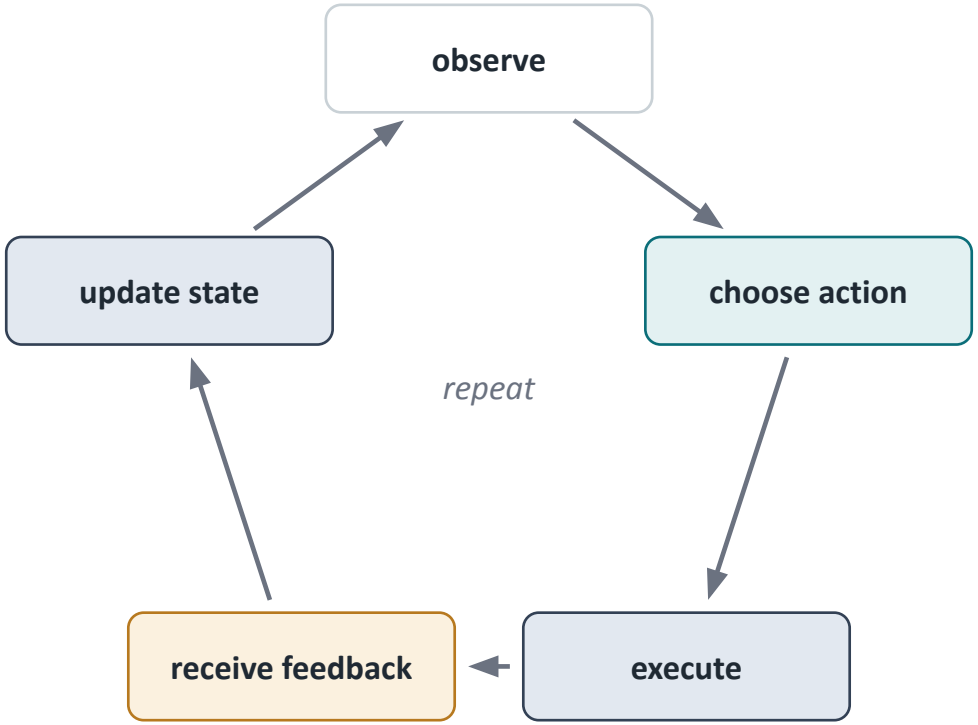
1. What is an agent?
2. Why are proof assistants interesting environments?
3. How SoTA evolve from 2020 to 2026?
4. What does test-time compute buy?
5. What remains after the kernel accepts?

A chatbot (pre-agentic) is normally one interaction

- the response may be long
- but there is no necessary external loop: no tools, no persistent state, no retry;
- whatever happens, happens inside one generation.

prompt → one response. Then the conversation waits for you.

An agent acts in a loop



- the loop runs until solved (or until the budget runs out);
- everything in this lecture is an instance of this loop.

Model versus harness

MODEL

- predicts the next useful action,
- given the goal, the current state, and the memory it is shown;
- stateless between calls it only sees what the harness sends.

HARNESS

- owns the tools and the environment handles;
- owns persistent state, memory, logs;
- owns budgets, retries, parallelism;
- owns the stopping conditions.

The model decides what to try. The harness decides what the model sees, what runs, and when it all stops.

Four ingredients beyond the model

environment — the thing acted upon

tools — actions beyond text (e.g. execute code)

memory / state — what survives between calls: attempts failed, files changed

control policy — what to try next and with what budget

All four live in the harness. The model is called by the control policy.

The smallest possible agent

```
state ← observe_environment()
while not solved and budget_remaining:
    action            ← model(goal, state, memory)
    feedback, state   ← environment.execute(action)
    memory            ← update(memory, action, feedback)
```

the model: one call per iteration

the environment: acts + answers

the harness: everything else

- every system in this lecture is this loop, with better proposals, better memory, better control.

Why long tasks fail

goal drift

quietly starts solving a different problem

repeated failures

retries the strategy that already failed

context overflow

the transcript no longer fits

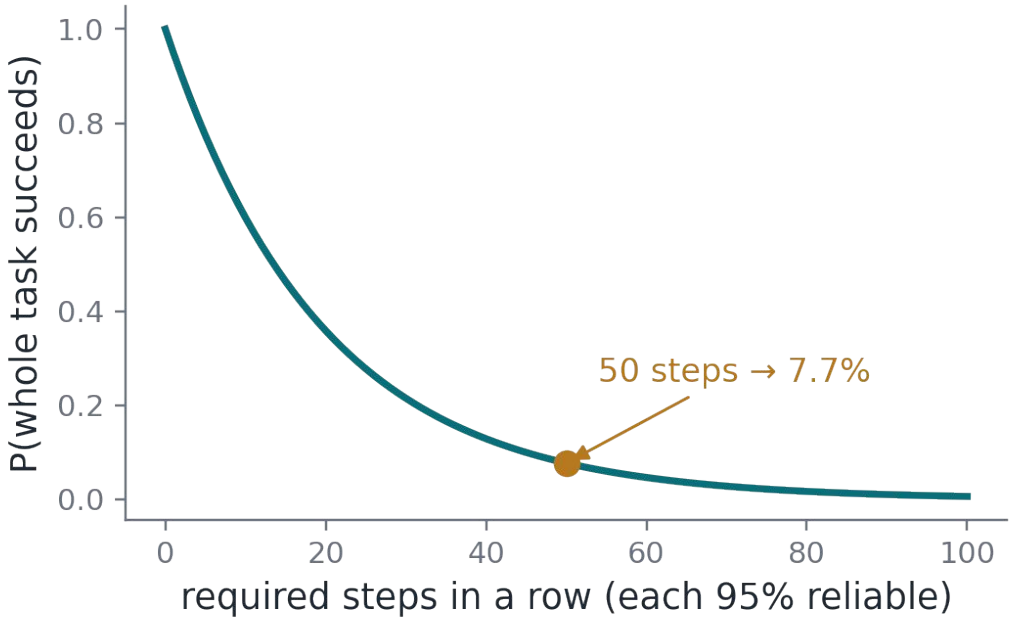
inconsistent edits

changes that contradict earlier changes

missed completion

solves the task

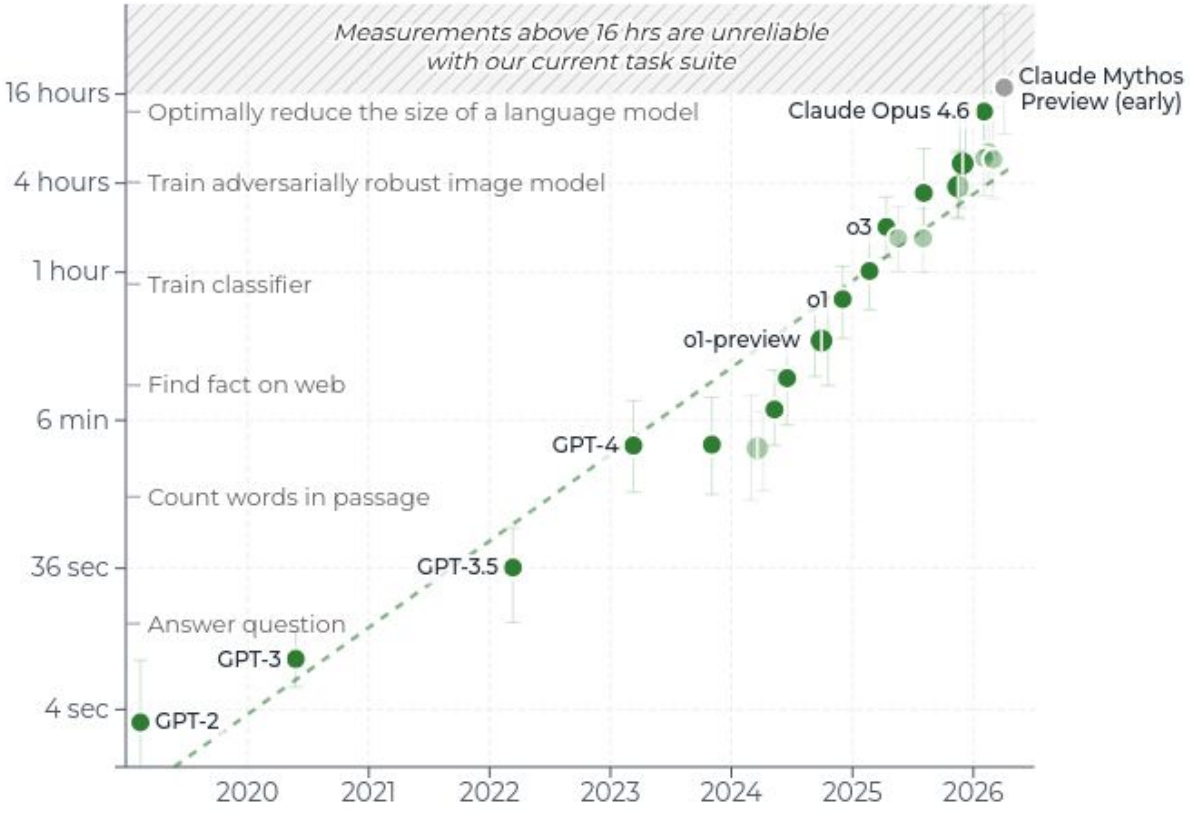
Local reliability does not imply global reliability



- **95% per step, 50 steps** → **$0.95^{50} \approx 7.7\%$** for the whole task;
- this is why “usually right per step” still feels broken on long tasks;

Either raise per-step reliability, or build a loop that catches and repairs failures.

How long a task can an agent complete?



The 50% time horizon

the human-expert duration of tasks that the model completes 50% of the time

- measured on software tasks, against human baseliners;
- doubling roughly every 7 months since 2020

Whatever the exact slope: the frontier of "how long an agent can stay coherent" is moving fast.

METR (2025), "Measuring AI Ability to Complete Long Software Tasks", Fig. 1 · arXiv:2503.14499

Proof assistants are clean environments

1	**machine-readable state**	goals, hypotheses, context
2	**executable actions**	tactics and terms are programs the environment runs
3	**immediate feedback**	each step answers with new goals or a structured error
4	**exact success criterion**	the kernel accepts, or it does not
5	**independently checkable artifact**	the finished proof re-checks without the agent

Almost no other agent environment offers all five at once.

The state the agent observes

```
a b : ℝ
h : a ^ 2 + b ^ 2 = 2
⊢ a * b ≤ 1
```

hypotheses above the line, the current goal after ⊢

- **local context and hypotheses**: what is known here;
- **one or more goals**: what remains to be proved;
- **imported libraries**: what exists to build on (Mathlib: >1.5M lines);
- **plus whatever the harness adds**: previous observations, retrieved lemmas, notes.

The actions the agent can take

generate a full script

one shot at the whole proof

apply a tactic

one step: intro, rw, cases …

search the library

find the lemma that already exists

add an intermediate lemma

have h₂ : … — split the problem

edit the statement

the statement itself can be wrong

The feedback the agent receives

accepted step	new (usually smaller) goals to work on
rejected step	a structured error: parse · elaboration · type mismatch · tactic failed · timeout
final success	the kernel accepts a complete proof term

A tiny interaction, end to end

informal

```
0 ≤ (a − b)²
⇒  2ab ≤ a² + b²
⇒  2ab ≤ 2
⇒  ab ≤ 1
```

formal (Lean 4 + Mathlib)

```
theorem square_sum_bound (a b : ℝ)
    (h : a ^ 2 + b ^ 2 = 2) :
    a * b ≤ 1 := by
  nlinarith [sq_nonneg (a - b)]
```

- **`nlinarith`** is automation
- the lemma sq_nonneg (a - b) is exactly the informal first line.

Three possible granularities

whole proof

one generation = one complete attempt

- cheap to parallelize
- no feedback until the end

tactic by tactic

one step per call, observe after each

- feedback at every step
- many calls; a search tree appears

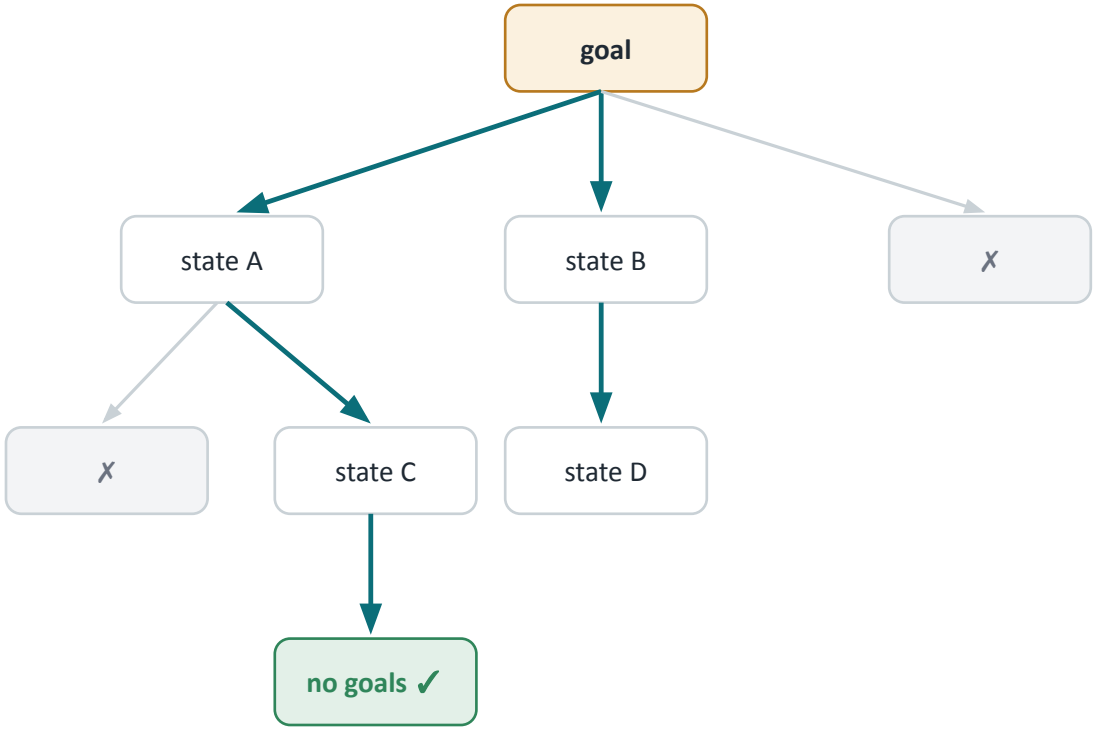
variable granularity

the agent chooses: prove, search, plan, decompose

- matches modern systems
- needs a real control policy

The granularity is part of the harness design

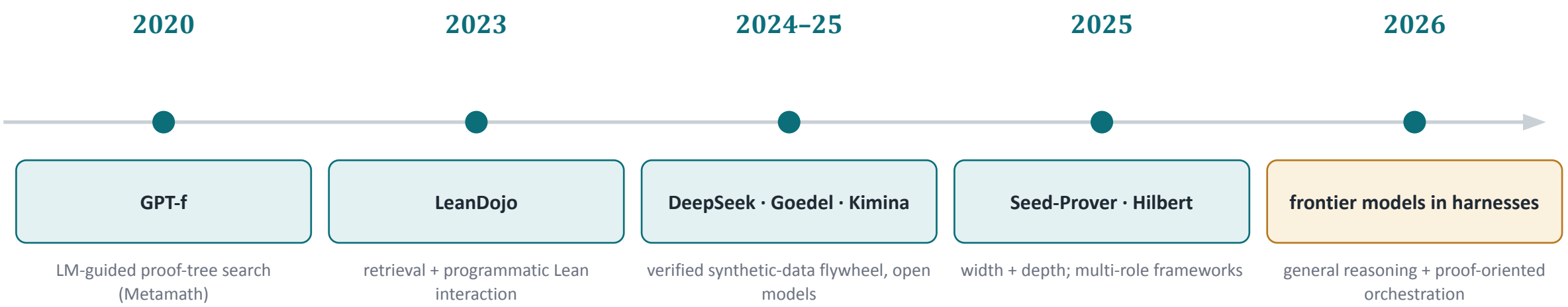
Formal proving is search



✗ = rejected by the verifier · teal path = actions that survived · green = proof found

- **nodes** are proof states; **edges** are actions;
- the verifier prunes edges for free, rejections cost nothing but compute;
- a branch reaching “no goals” is a complete proof;
- **search needs what the harness owns:** a frontier, memory of failures, an expansion policy, a budget.

A compressed and incomplete history: 2020 → 2026



GPT-f (2020): training

CommonCrawl

Table 1: Mix and source of data involved in the *WebMath* dataset.

Dataset	Size	Mix
Github	23 GB	33%
arXiv Math	10 GB	33%
Math StackExchange	2 GB	33%

Metamath proofs

GPT-f (2020): generate the next proof step

the model's task

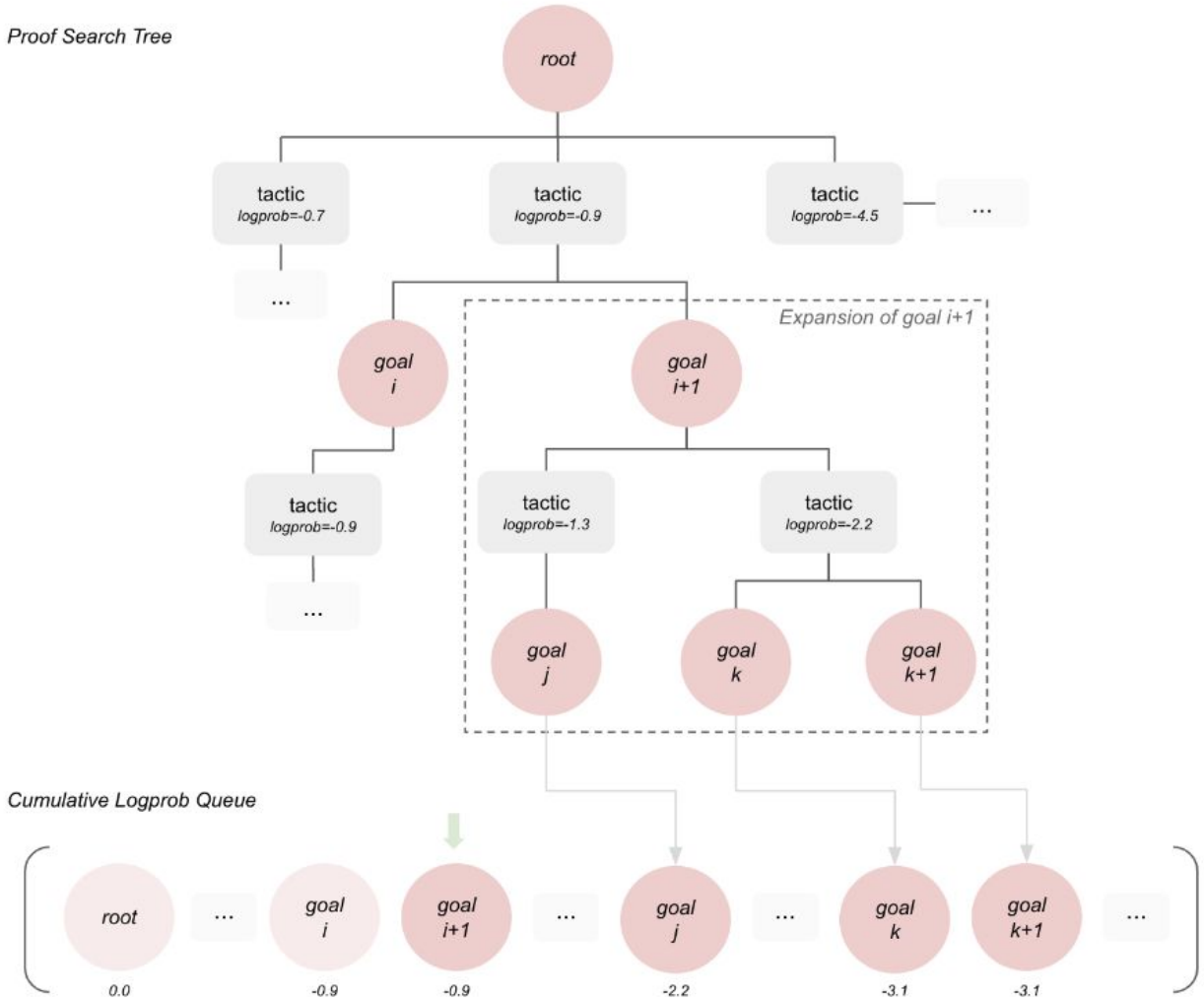
```
GOAL  ⊢ ( 2 + 2 ) = 4
PROOFSTEP ?
```

```
eqcomi  ·  2p2e4  ·  …  (candidate steps, with
scores)
```

- a decoder-only transformer: lecture 1's architecture, unchanged;
- input: a Metamath goal; output: a candidate substitution / proof step;

Polu & Sutskever (2020), "Generative Language Modeling for Automated Theorem Proving" · arXiv:2009.03393

GPT-f used a search tree



- the queue is sorted by cumulative log-probability — the model's own confidence guides the search;
- rejected candidates are simply dropped: the verifier prunes for free;
- **budgets everywhere:** e candidates per goal, depth d, and a independent search attempts (a = 32 for the final evaluation).

Polu & Sutskever (2020), Fig. 1 · arXiv:2009.03393

Broad pretraining transfers to formal proving

Table 7: Performance for various model sizes and pre-training datasets.

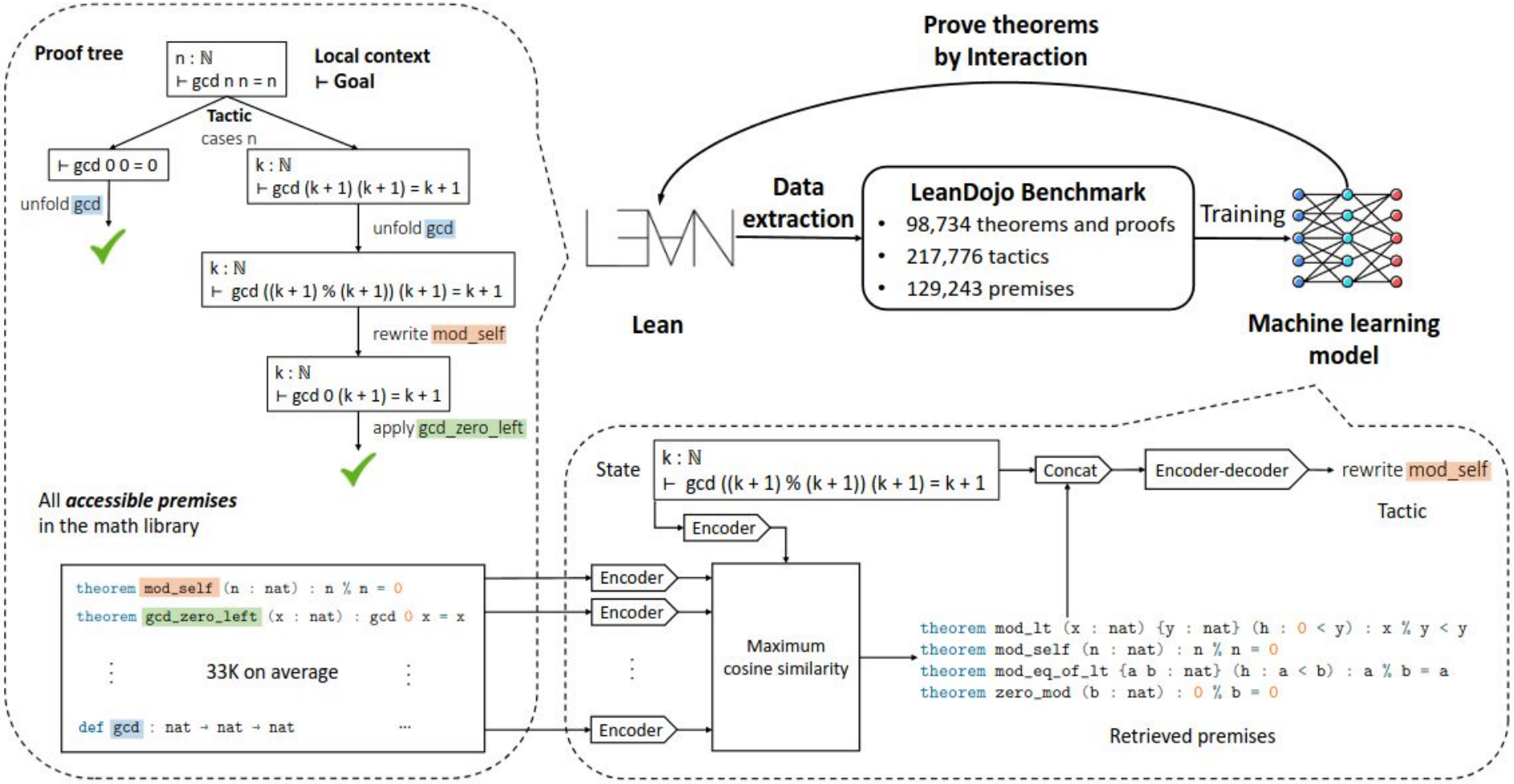
Model	Performance	Perplexity	# Tokens
160m *from scratch*	28.96%	1.041	18B
160m *CommonCrawl*	32.34%	1.030	16B
160m *Github*	33.61%	1.030	16B
160m *WebMath*	34.79%	1.029	16B
700m *from scratch*	31.58%	1.040	18B
700m *CommonCrawl*	39.61%	1.026	15B
700m *Github*	41.55%	1.025	15B
700m *WebMath*	**42.56%**	**1.024**	15B

- same architecture, same formal fine-tuning, only the pretraining differs;
- +11 points from general pretraining before any formal data;
- **the lesson:** broad text, code and math pretraining transfers into formal proving, already in 2020.

Model	Performance	Gain	Main ablation
MetaGen-IL [25]	21.16%		Baseline and state of the art.
160m (ours)	28.96%	+7.8%	Use of Transformers.
700m (ours)	31.58%	+2.5%	Increase in parameters count.
700m *WebMath* (ours)	42.56%	+10.9%	Pre-training.
700m *policy+value* (ours)	47.21%	+4.6%	Iterated learned value function.
700m *policy+value* $a = 32$ (ours)	56.50%	+9.2%	Increased test-time compute.

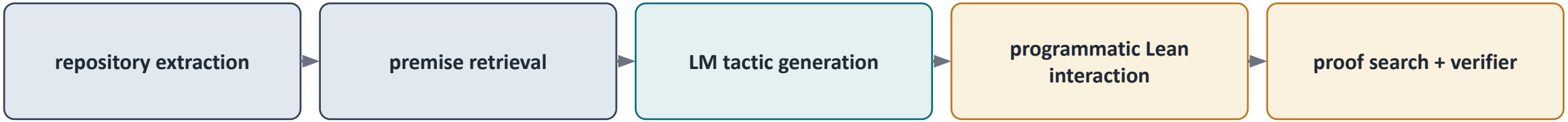
Polu & Sutskever (2020), Table 7 · arXiv:2009.03393

LeanDojo (2023): connect the model to the library and to Lean



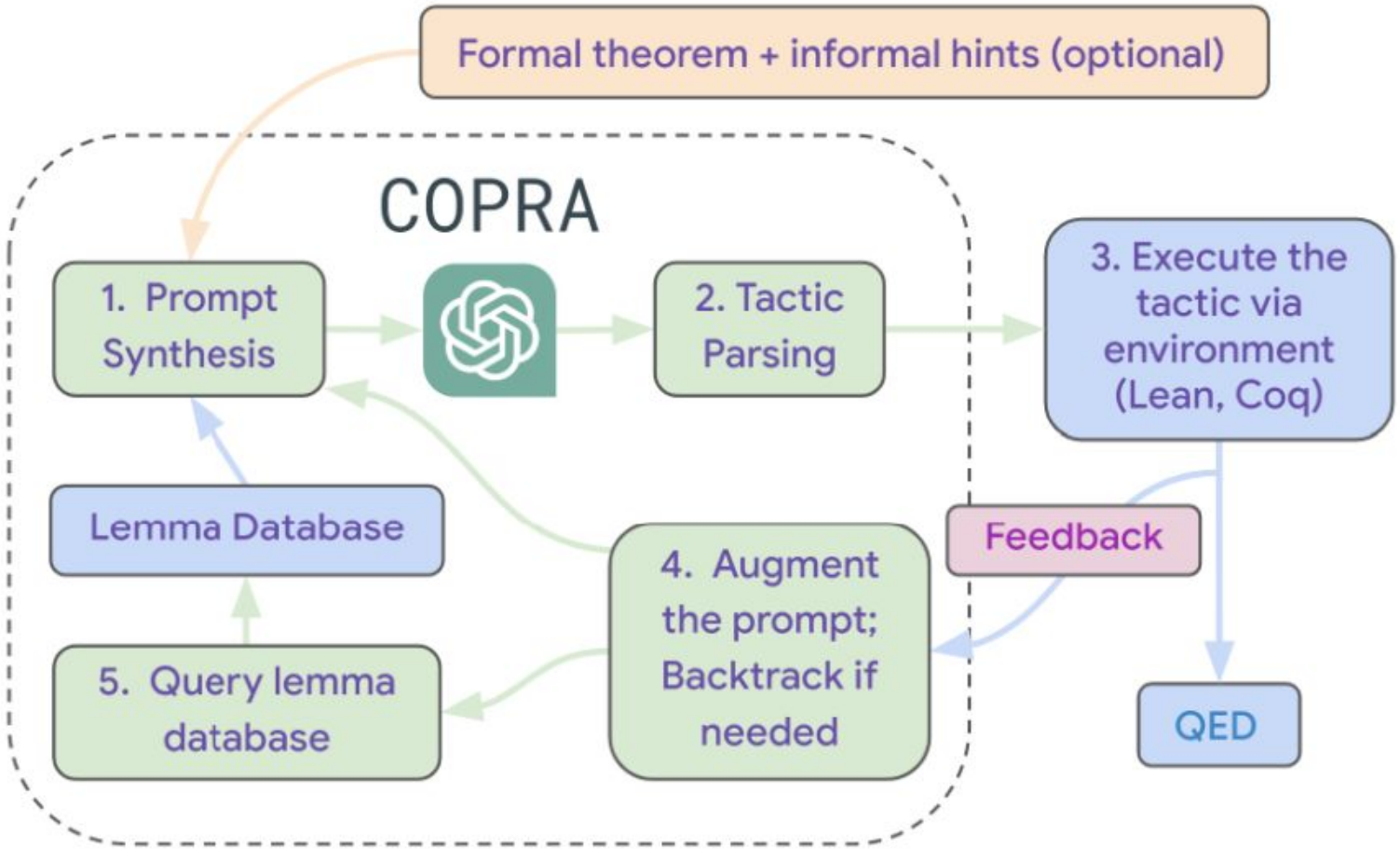
"LeanDojo: Theorem Proving with Retrieval-Augmented Language Models", arXiv:2306.15626

The LeanDojo lesson



2023: every harness ingredient of section 1 is already on the table.

COPRA (2023): LLM + harness



At first, miniF2F was genuinely difficult

Lean GPT-f / PACT

24.6 %

LeanDojo (ReProver)

25.0 %

COPRA

30.7 %

Zheng, Han, Polu (2021), Table 3 · arXiv:2109.00110

The central training bottleneck: formal data

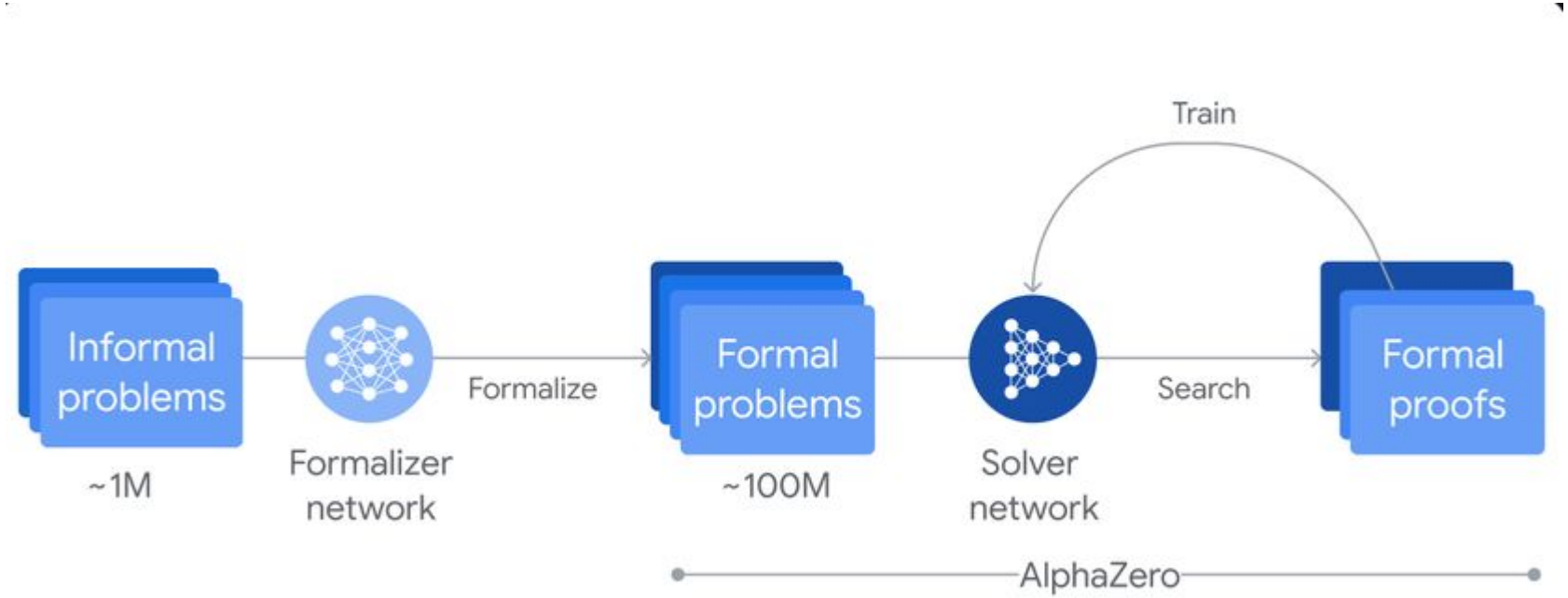
informal mathematics

- millions of papers, textbooks, solutions, forum posts;
- noisy, but everywhere;
- this is what pretraining already ate.

correct formal proofs

- scarce: libraries total a few million lines;
- specialized: written by a small community;
- version-bound: proofs break as the library moves.

A verifier creates a data flywheel (AlphaProof)

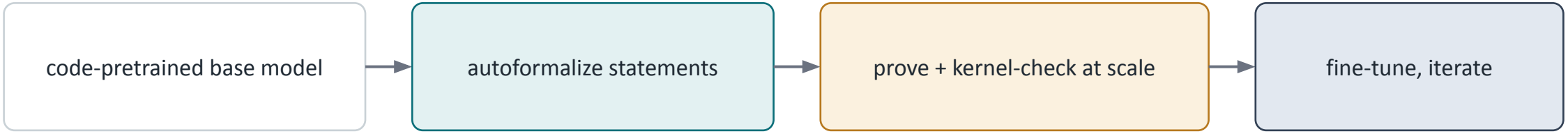


July 25, 2024 Science

AI achieves silver-medal standard solving International Mathematical Olympiad problems

AlphaProof and AlphaGeometry teams

DeepSeek-Prover (2024): formal training data at scale



- the base model matters: code pretraining transfers (GPT-f's lesson, again);
- statement quality is the weak point: degenerate or unfaithful statements poison the flywheel;
- each iteration: stronger prover → more verified proofs → better training set.

Xin et al. (2024), DeepSeek-Prover · arXiv:2405.14333

Goedel-Prover (2025): the flywheel, industrialized

Goedel-Pset

1.64 M

generated formal statements

kernel-verified

800 K+

proved statements used in training

process

iterative

each expert round feeds the next

- statements: translated and generated, then filtered;
- proofs: only kernel-accepted ones enter the training set;
- **the point is the process**.

Lin et al. (2025), "Goedel-Prover" · arXiv:2502.07640

miniF2F: a common formal-mathematics test

488

problems: 244 valid + 244 test

3 sources

AIME · AMC · IMO

multi-system

Metamath · Lean · Isabelle · HOL Light

- statements formalized in parallel across proof assistants
- Olympiad problems: hard, but self-contained

Zheng, Han, Polu (2021), "miniF2F: a cross-system benchmark for formal Olympiad-level mathematics" · arXiv:2109.00110

What a miniF2F entry looks like

informal (AMC-style)

What is the units digit of 13^{2021}?

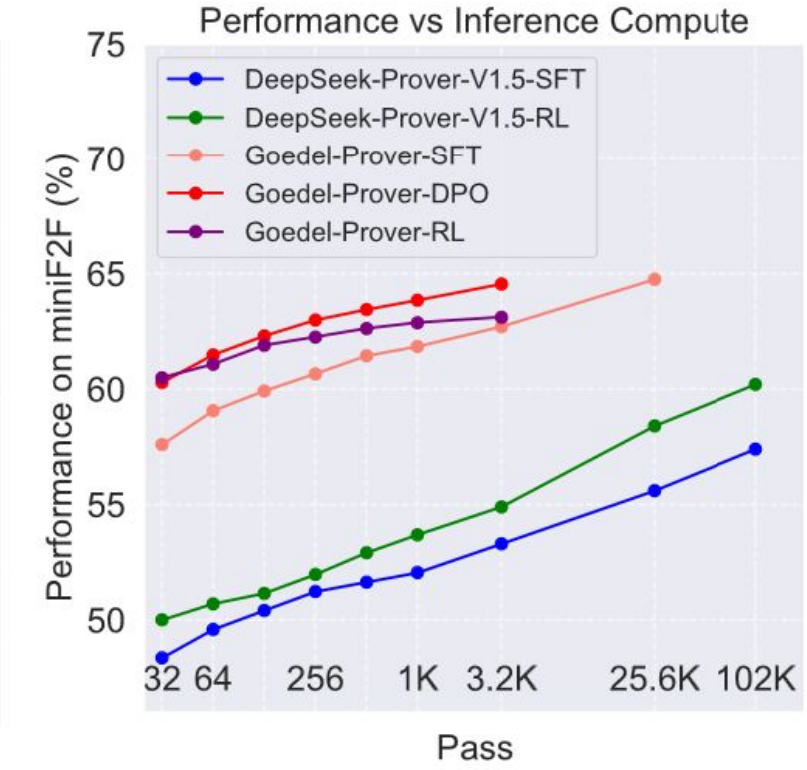
formal (Lean)

```
theorem units_digit :
    13 ^ 2021 % 10 = 3 := by
  norm_num
```

any script the kernel accepts counts

Statement style after miniF2F (arXiv:2109.00110); this instance simplified for the slide

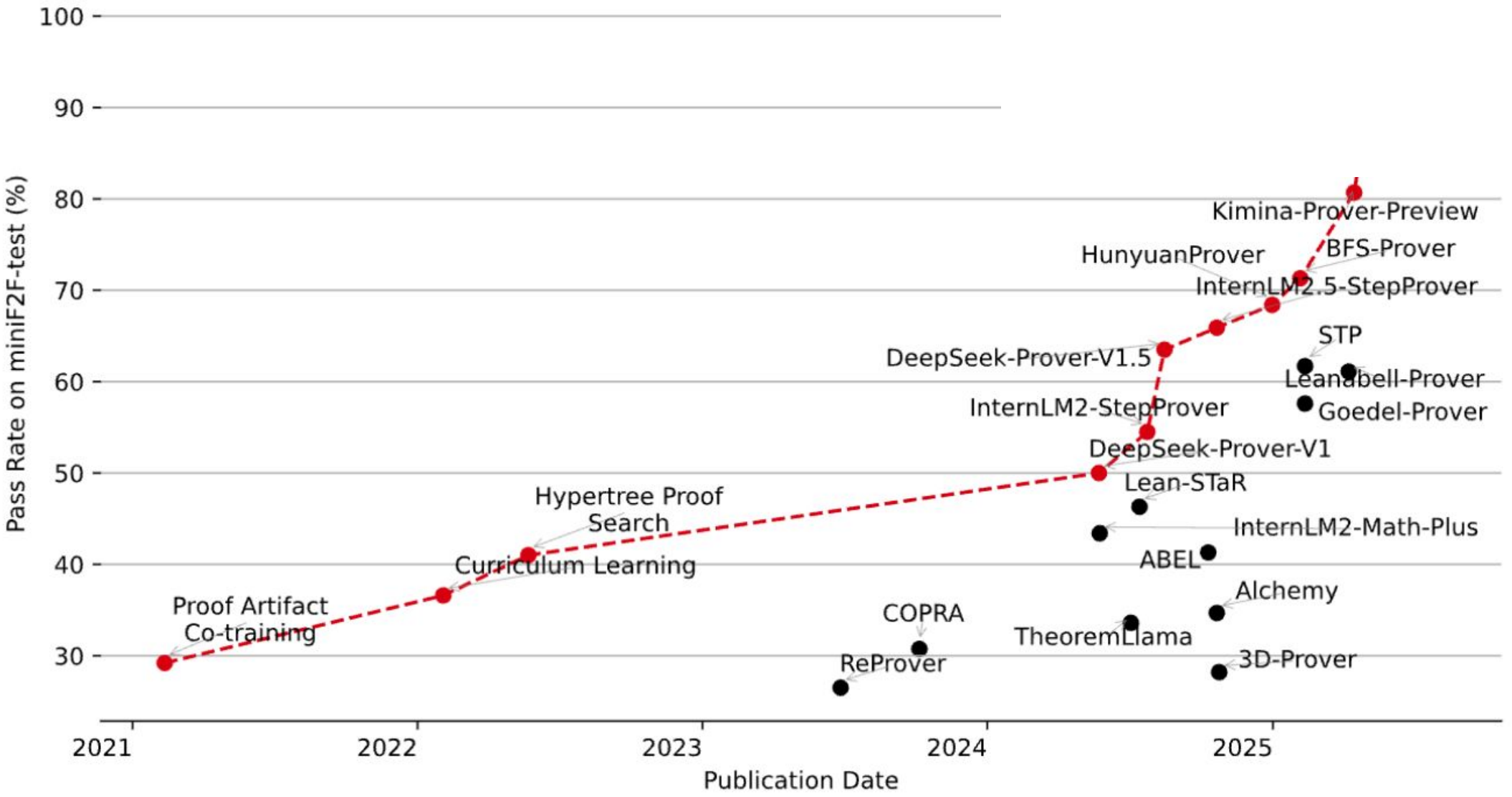
When data is too expensive increase test-time budget



- **57.6% miniF2F at pass@32** (then state of the art for open models);
- **7 PutnamBench problems at pass@512** — then #1 on that leaderboard;
- still rising at aggregate budgets up to 25,600 samples;
- log x-axis

After Lin et al. (2025), Fig. 1 · arXiv:2502.07640

MiniF2F result



PutnamBench: a harder step

total

1,724

hand-built formalizations

Lean 4

672

problems

Isabelle

640

problems

Rocq / Coq

412

problems

- undergraduate competition mathematics (Putnam).
- hand-formalized in three systems; statements only.

Tsoukalas et al., PutnamBench · trishullab.github.io/PutnamBench (counts retrieved Aug 2026)

What a PutnamBench problem looks like

informal (Putnam 1988 B1)

A composite is a product ab with a, b ≥ 2.
Show: every composite n can be written as
n = xy + xz + yz + 1, with x, y, z positive integers.

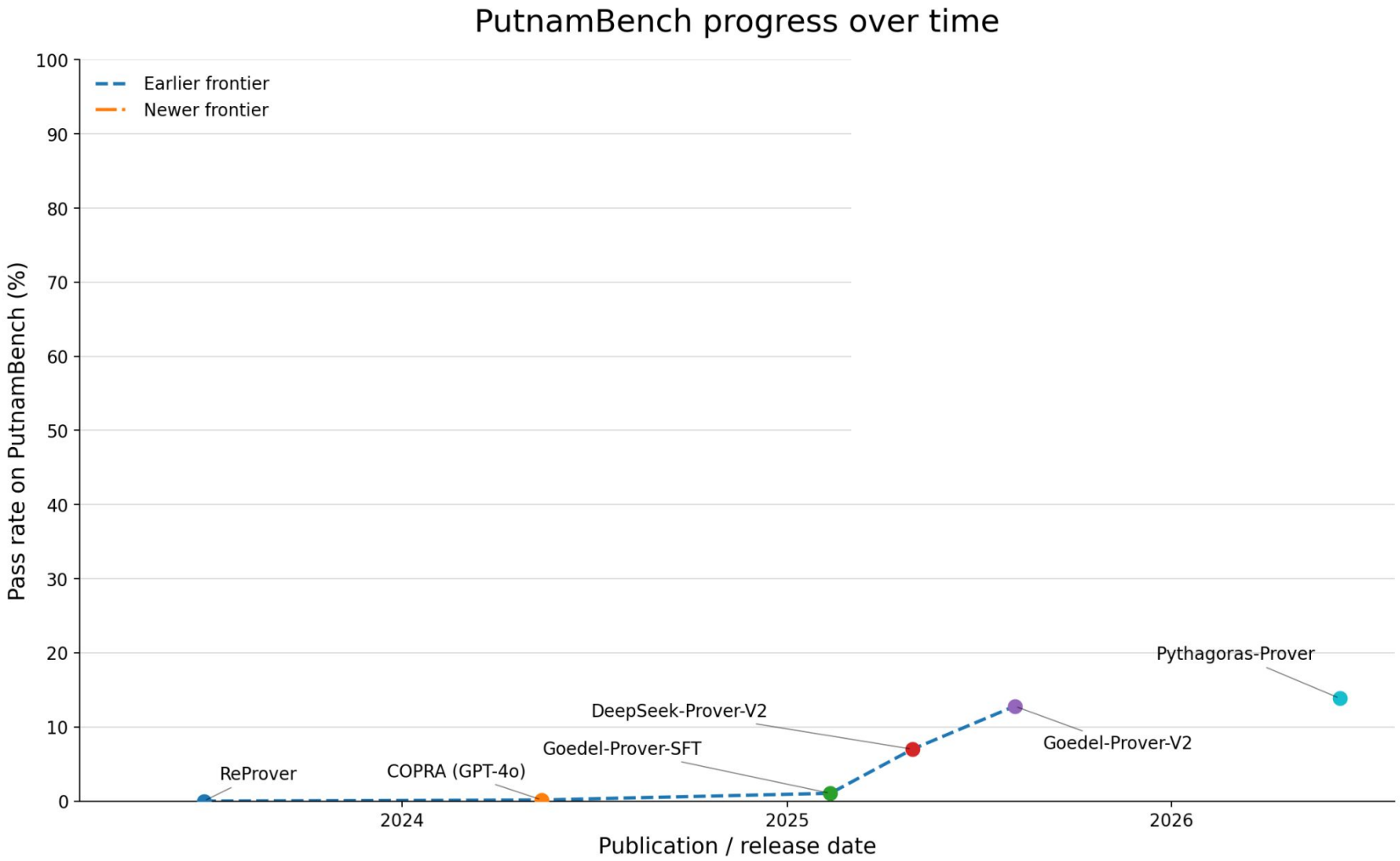
formal signature (Lean)

```
theorem putnam_1988_b1
    (n : ℤ) (h : IsComposite n) :
    ∃ x y z : ℤ, 0 < x ∧ 0 < y ∧ 0 < z ∧
      n = x*y + x*z + y*z + 1 := by
  sorry
```

first idea: take z = 1 — then xy + x + y + 1 = (x+1)(y+1), and n = ab does the rest

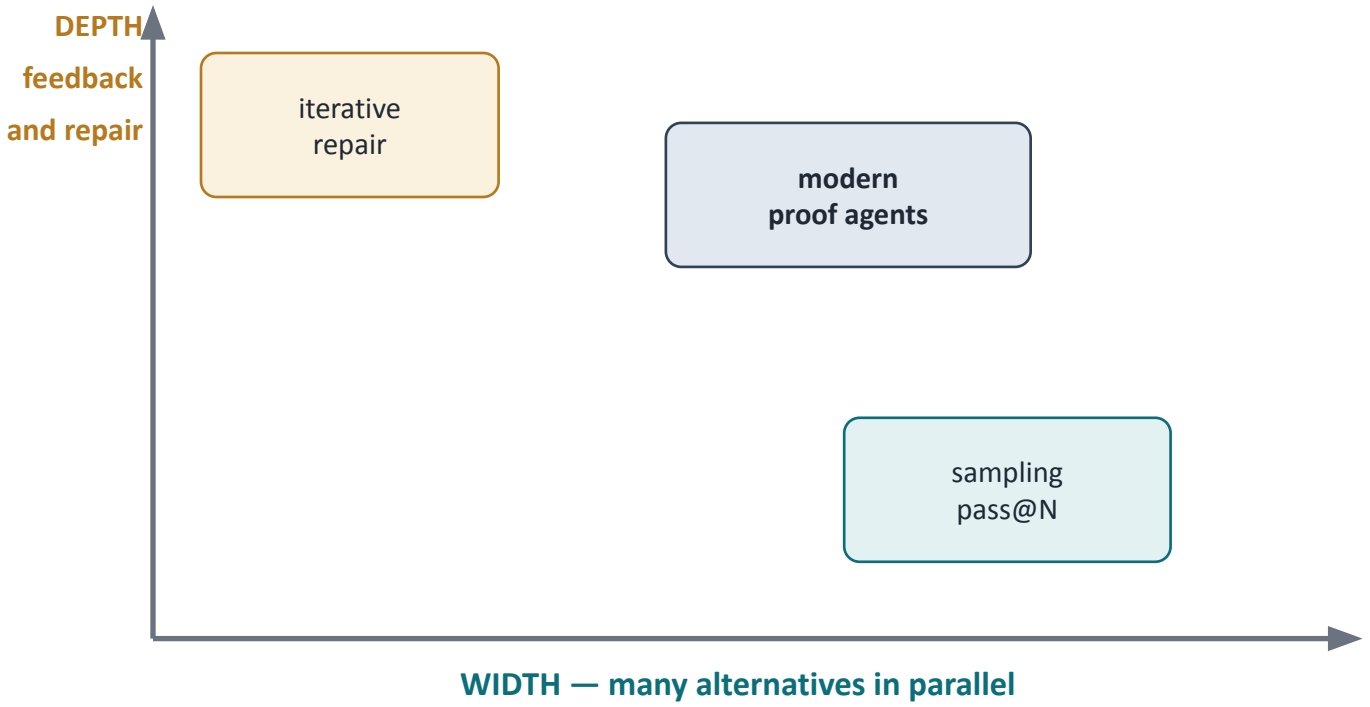
PutnamBench project page example (1988 B1), signature simplified for the slide

Putnam bench result



Larger context, longer inference

Two dimensions of test-time computation



- **width:** many alternatives, cheap, parallel, independent-ish;
- **depth:** one line of attack, driven by feedback and repair;
- **modern systems move diagonally**: both at once.

Hilbert

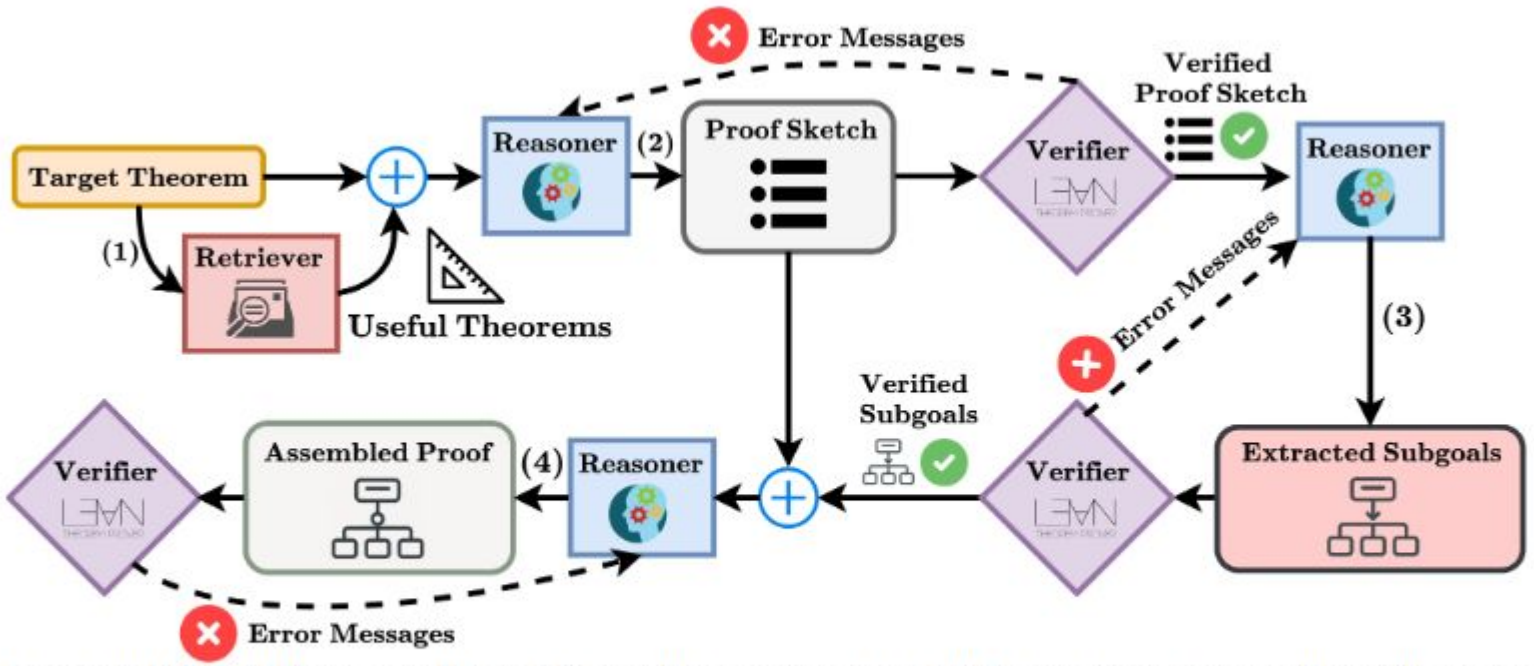
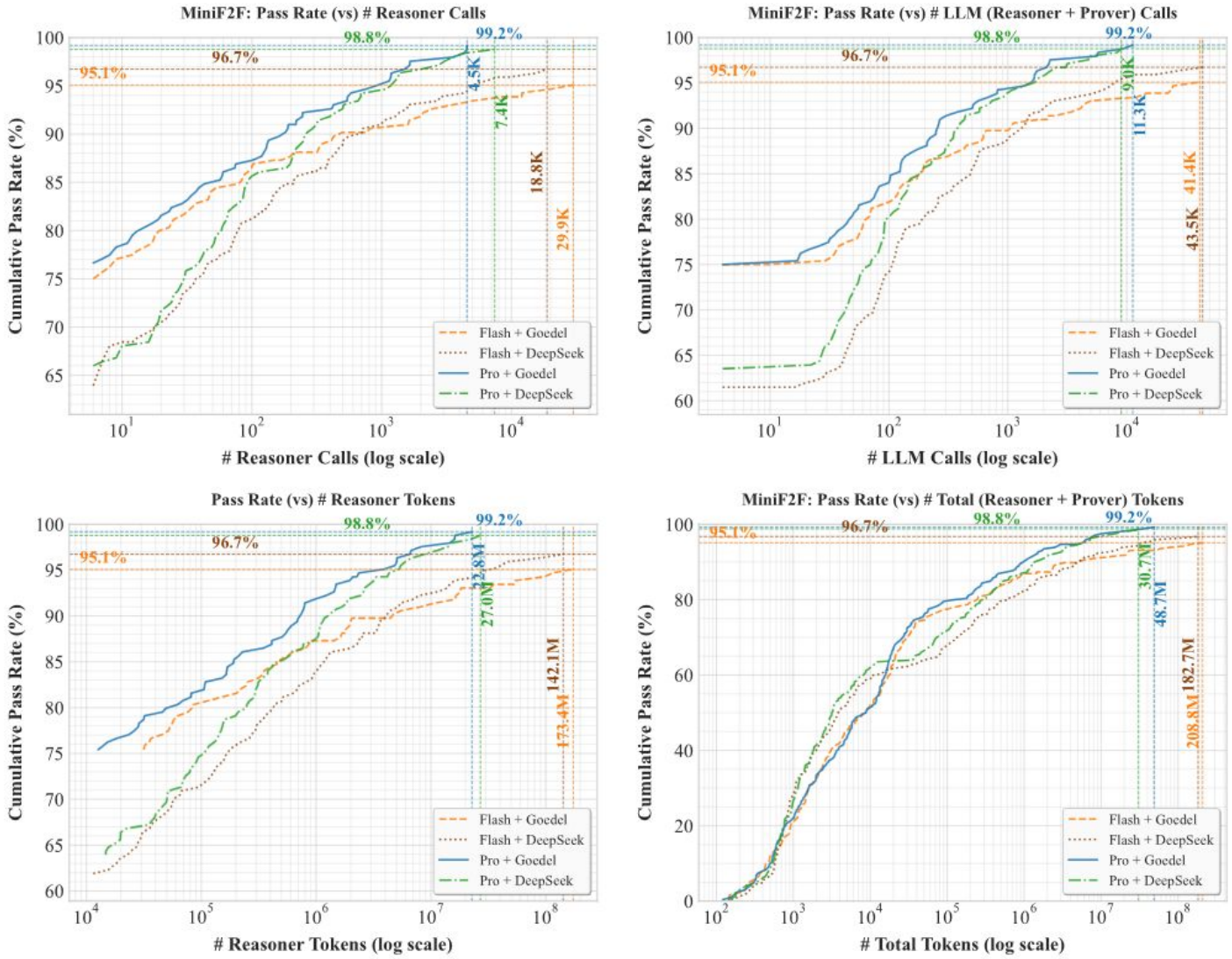


Figure 2: **Subgoal Decomposition:** Given a theorem statement, HILBERT: (1) retrieves relevant theorems from Mathlib using semantic search, (2) generates a formal proof sketch with subgoals marked as `have` statements with `sorry` placeholders, (3) extracts these subgoals as independent theorem statements, and (4) assembles the proof by replacing `sorry` placeholders with calls to the subgoal theorems. Verifiers ensure correctness at each stage. The error correction loops are indicated by dotted lines.

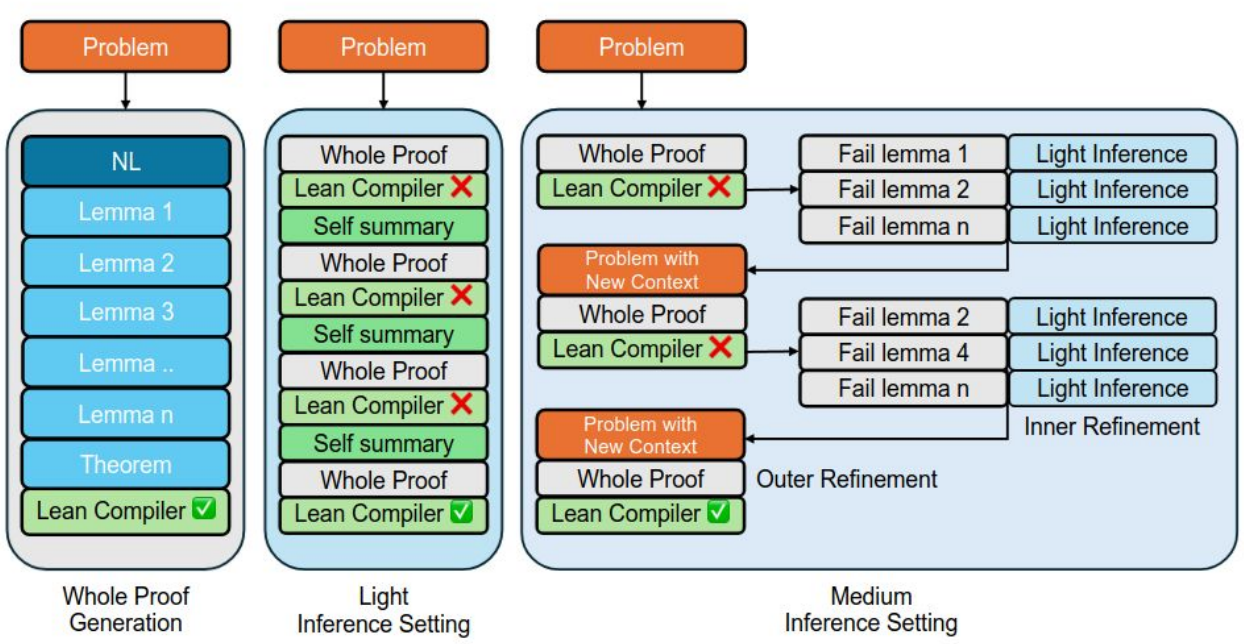
Seed-Prover 1.5 (2025) · arXiv:2512.17260

Hilbert



Seed-Prover 1.5 (2025) · arXiv:2512.17260

Seed-Prover 1.5: width and depth, trained together



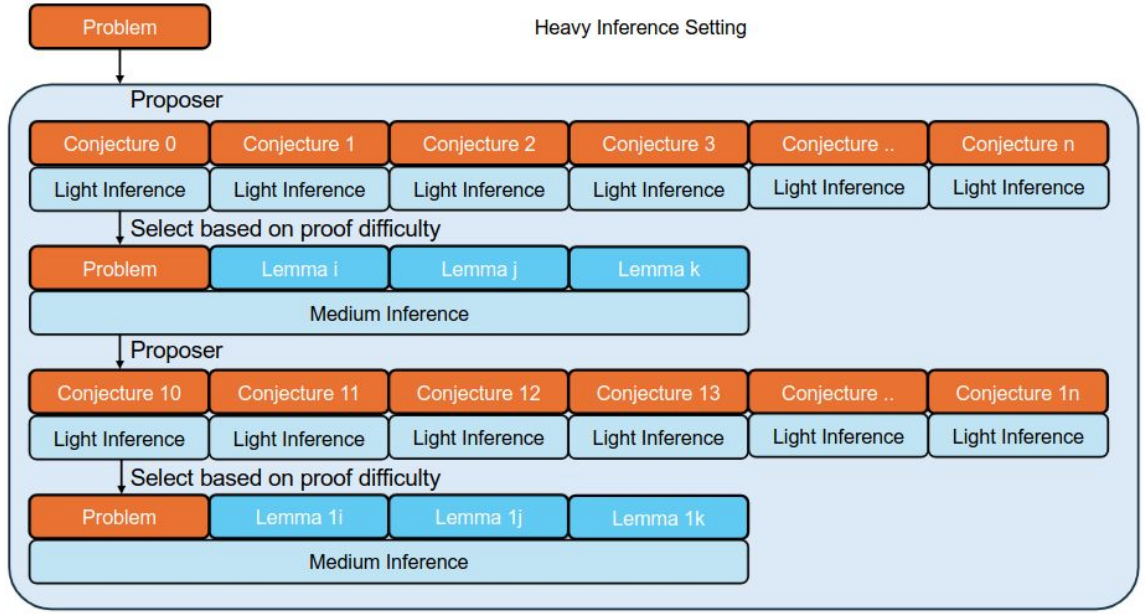
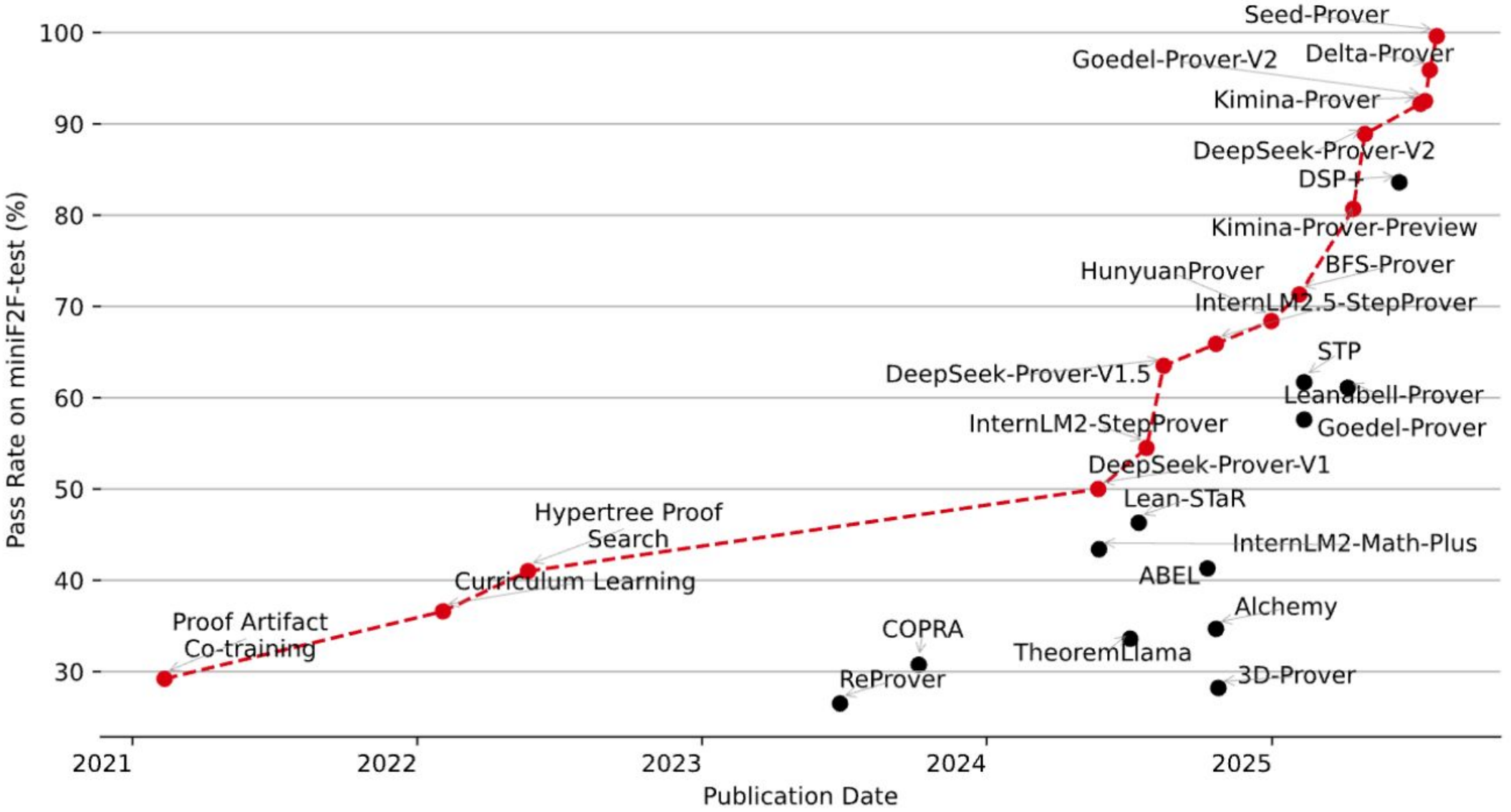


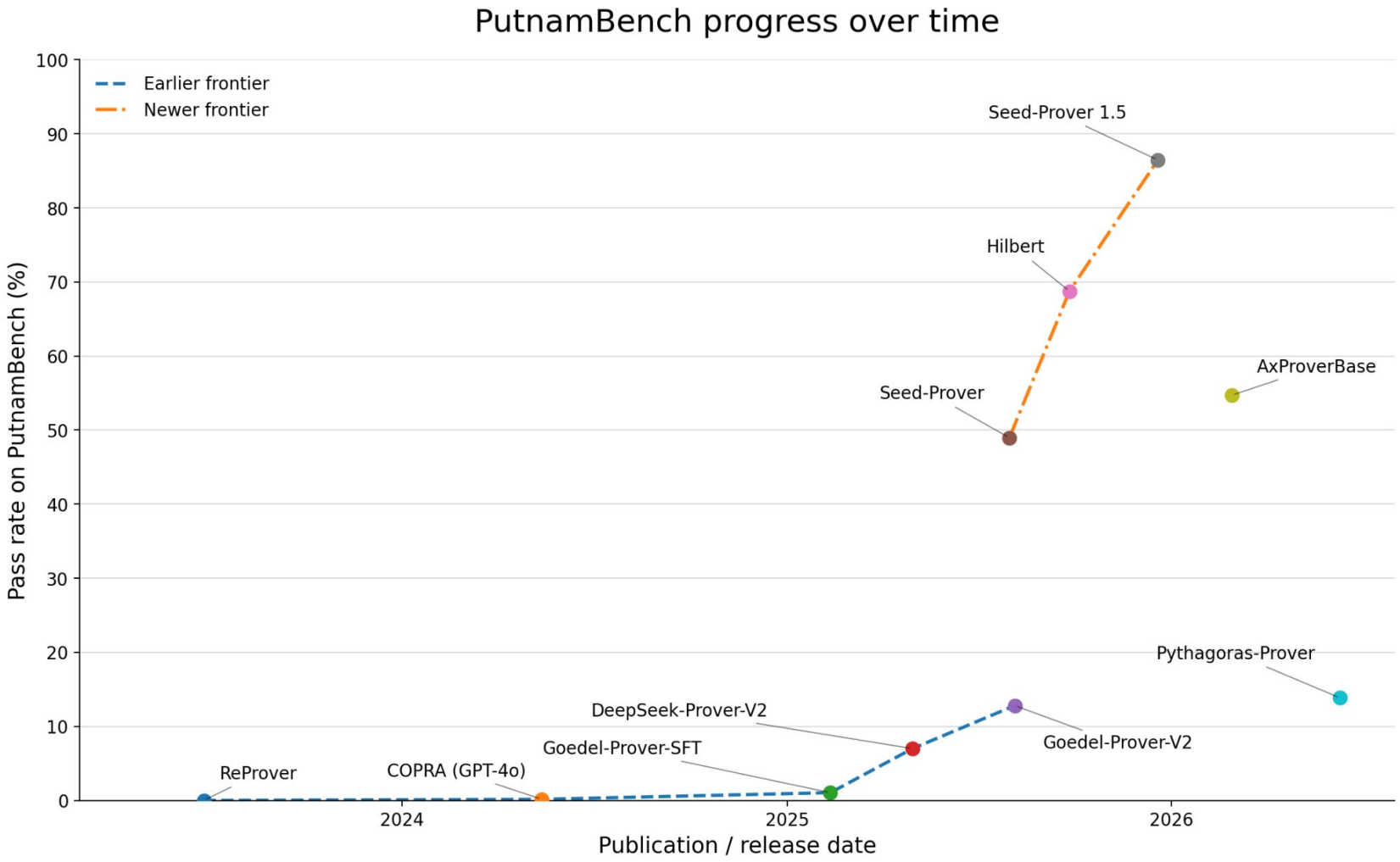
Figure 3 The workflows of single-pass whole proof generation, light, and medium inference settings.

Seed-Prover 1.5 (2025) · arXiv:2512.17260

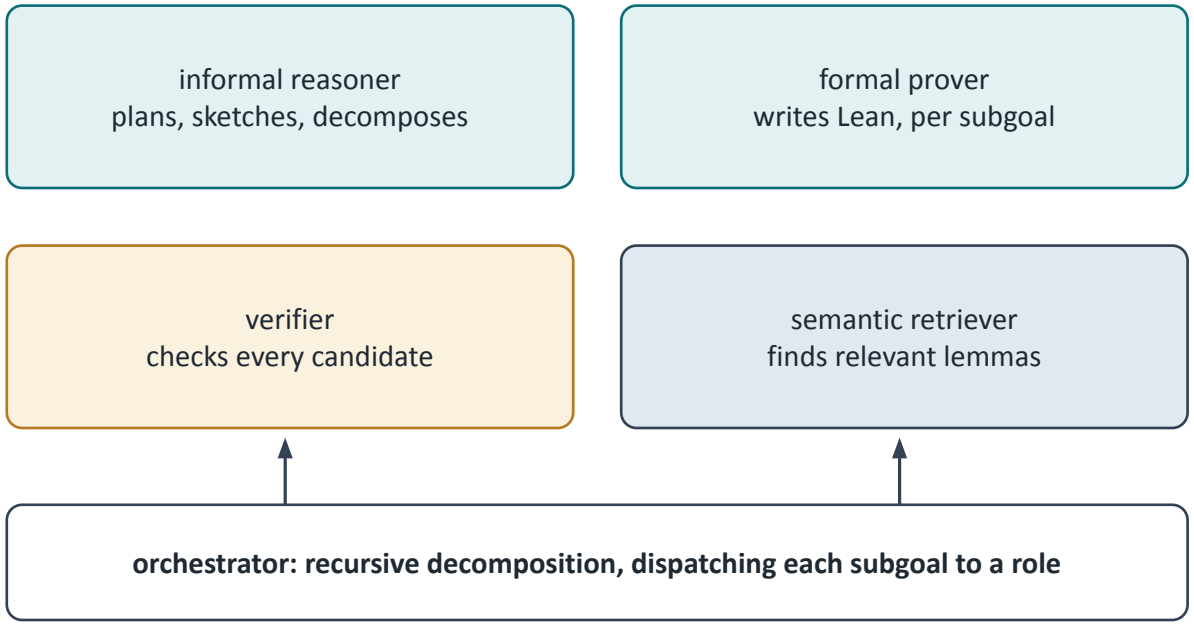
MiniF2F result



Putnam bench result



Hilbert is a framework, not a model



- recursive decomposition: hard goal ⟶ lemmas ⟶ sub-lemmas, each dispatched to a role;
- the roles can run different models general reasoner + specialist prover;

Hilbert (2025) · arXiv:2509.22819

Another example Rocq-MCP on Putnam 2025

subagents	tokens	tool calls
141	≈ 1.9 B	12,427

active compute	wall clock	proved
17.7 h	51.6 h	10 / 12

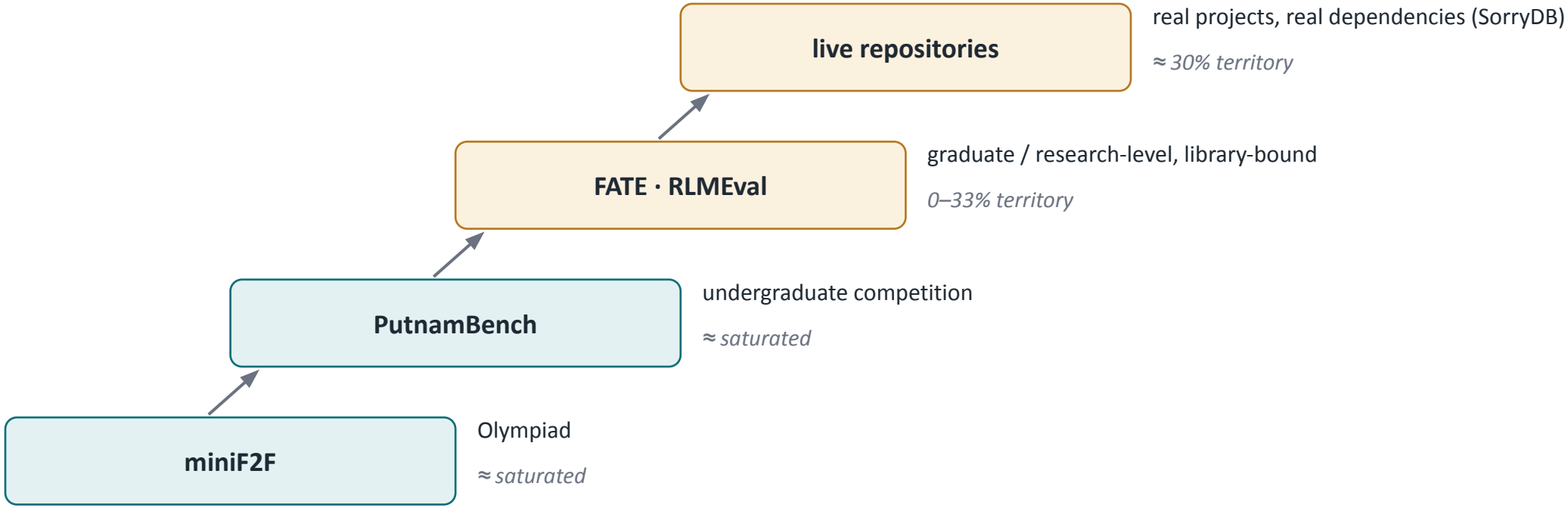
Putnam 2025 in Rocq · Claude Opus 4.6 + Rocq-MCP · 5,542 lines of accepted Rocq · A5, B6 unsolved

A striking systems experiment — not a standard benchmark budget.

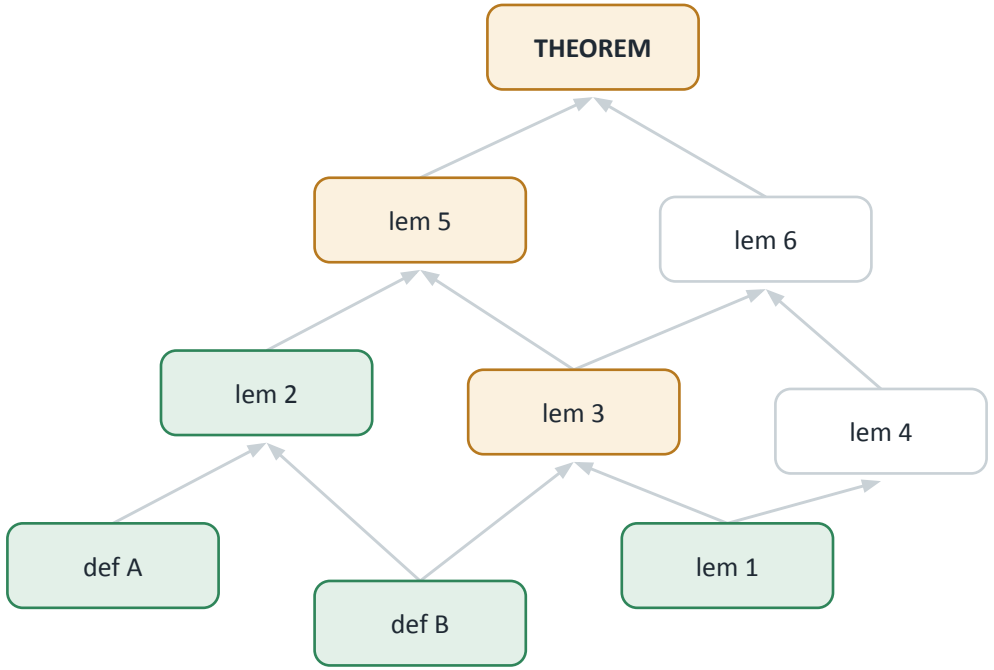
Baudart, Lelarge, Stérin, Viennot (2026) · arXiv:2603.20405

Next benchmarks

The benchmark ladder keeps moving



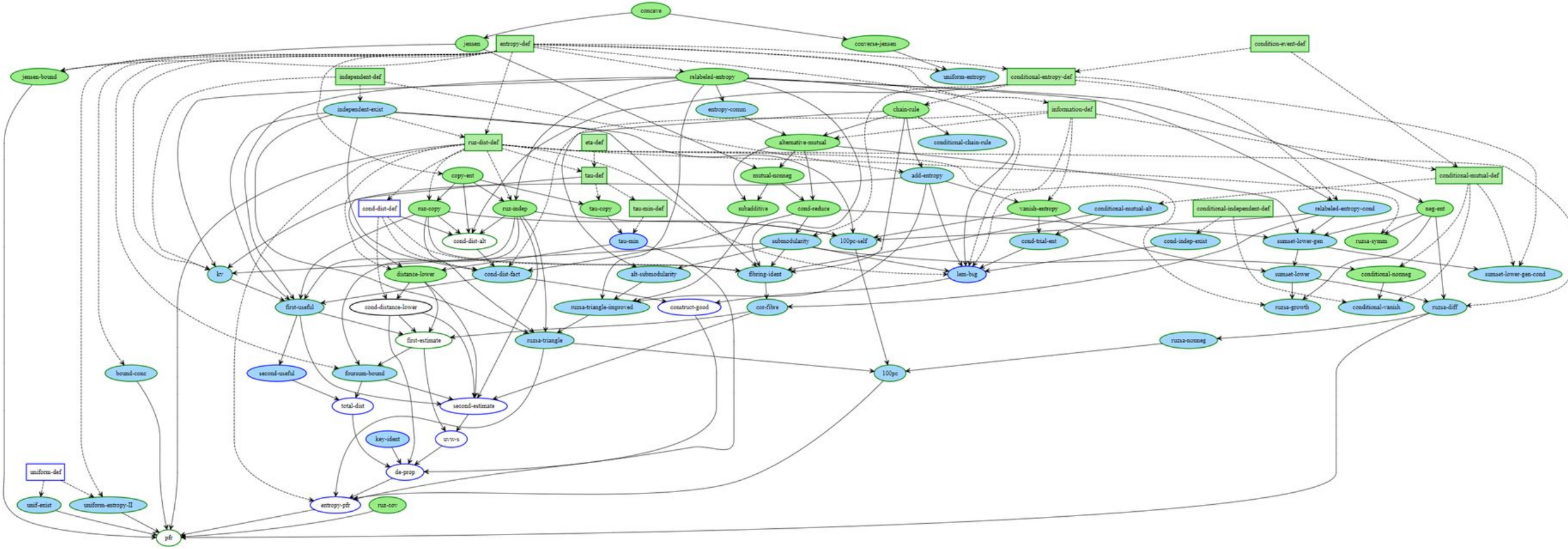
A research project is a dependency graph



green = formalized · amber = stated, unproven · white = informal only

- one theorem = one node in a living graph;
- it needs project-local definitions and dozens of auxiliary lemmas;

Research mathematics remains difficult



RLMEval (2025), Table 1 · arXiv:2510.25427

Research-level blueprints: RLMEval

RLMEval

613

theorems from real projects

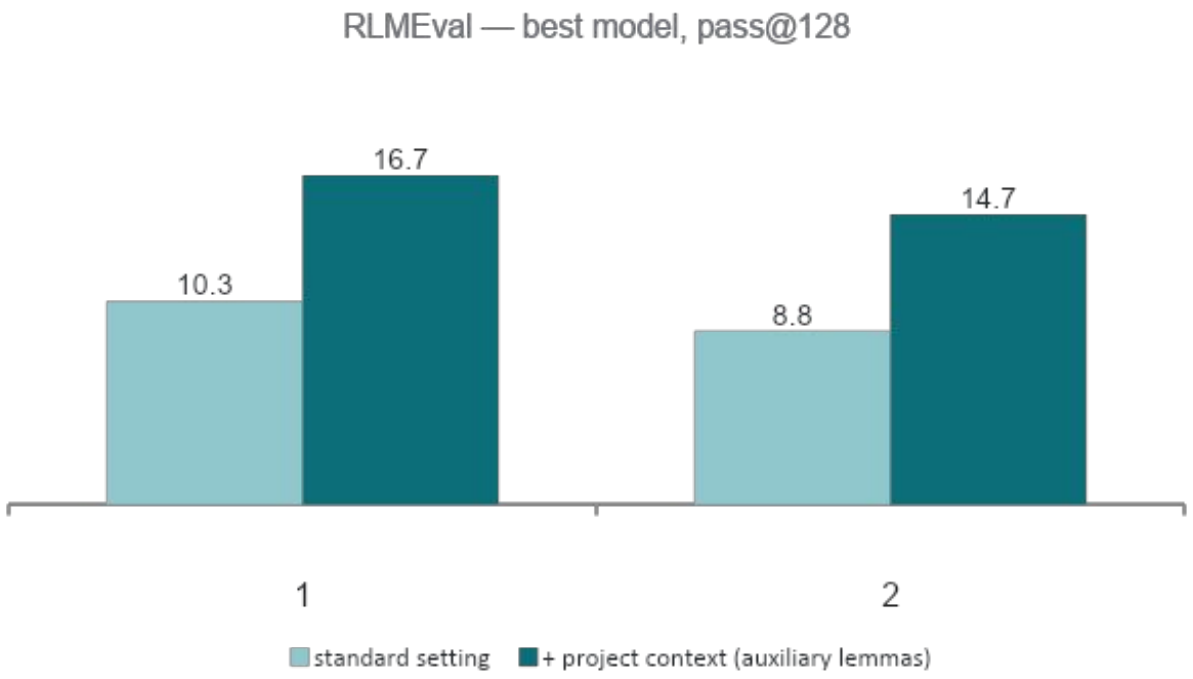
Carleson · PFR · FLT · FLT3 · PrimeNumberTheoremAnd · TLB

six live Lean Blueprint formalization projects

- **the target theorem sits in a dependency graph**
- statements reference the project's own vocabulary, not just Mathlib;
- this is the benchmark whose shape matches formalization work.

Poiroux et al. (2025), RLMEval · arXiv:2510.25427

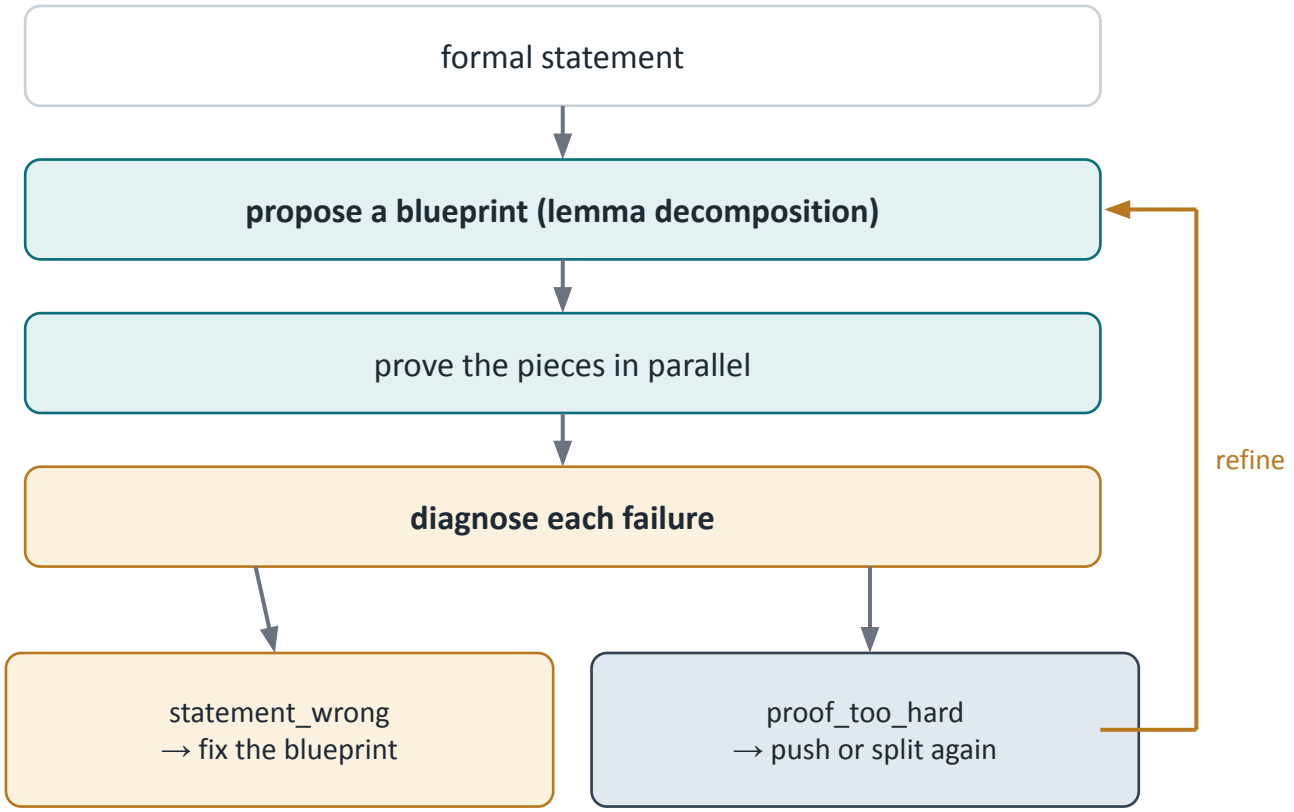
Research mathematics remains difficult



- **10.3% pass@128**, against 99% on miniF2F, same era;
- the agent that knows the project beats the model that does not.

RLMEval (2025), Table 1 · arXiv:2510.25427

Goedel-Architect: repair the blueprint, not only the proof



Redrawn after Goedel-Architect (2026), Fig. 1 · arXiv:2606.06468

Live repository tasks are different: SorryDB

tasks

5,663

sorries from 78 repos

evaluated

1,000

most recent snapshot

best single config

30.3%

agentic · ≤16 iterations

the task: fill a hole in someone's real project;

Current limitations

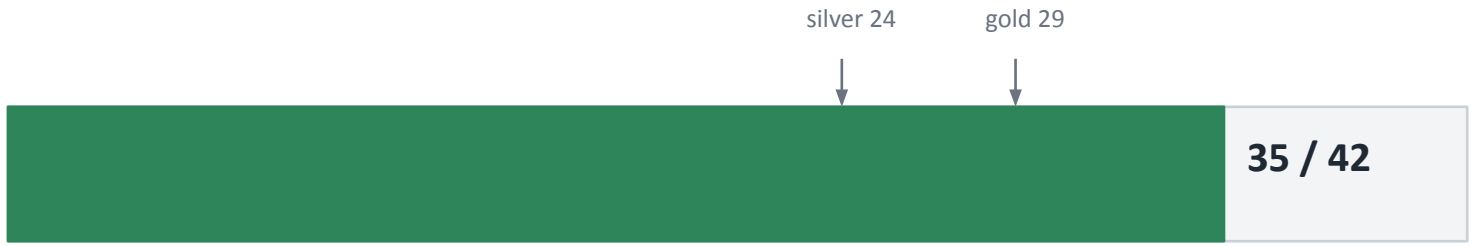
2024: AlphaProof reaches IMO silver level

IMO 2024 · 4 of 6 problems solved (2 algebra + 1 number theory by AlphaProof; geometry by AlphaGeometry 2 in 19 s)

- **label 1:** problems manually translated into formal language (Lean);
- **label 2:** computation ran up to three days per problem.
- **output:** kernel-checked formal proofs.

Google DeepMind (July 2024), "AI solves IMO problems at silver medal level" — official graphic on the blog

2025: natural-language performance reaches gold level



IMO 2025 · 5 of 6 problems, end-to-end in natural language, within the official 4.5-hour limit

- **no formal system anywhere:** official statements in, natural-language proofs out, graded by humans;
- **contest conditions on time**: 4.5 hours.

Google DeepMind (July 2025), Gemini Deep Think IMO announcement — official graphic on the blog

The limits of depth and width

diminishing returns

log-axis: 10× compute, few points

cost

GPU-hours scale linearly with number of pass, quadratically with the context size

duplicated effort

the same dead ends, thousands of times

loophole exploitation

finds reward hacks

Agents fail in agentic ways

sticky false beliefs

one wrong assumption may survives every iteration

edit loops

A→B→A→B between two versions

loophole proofs

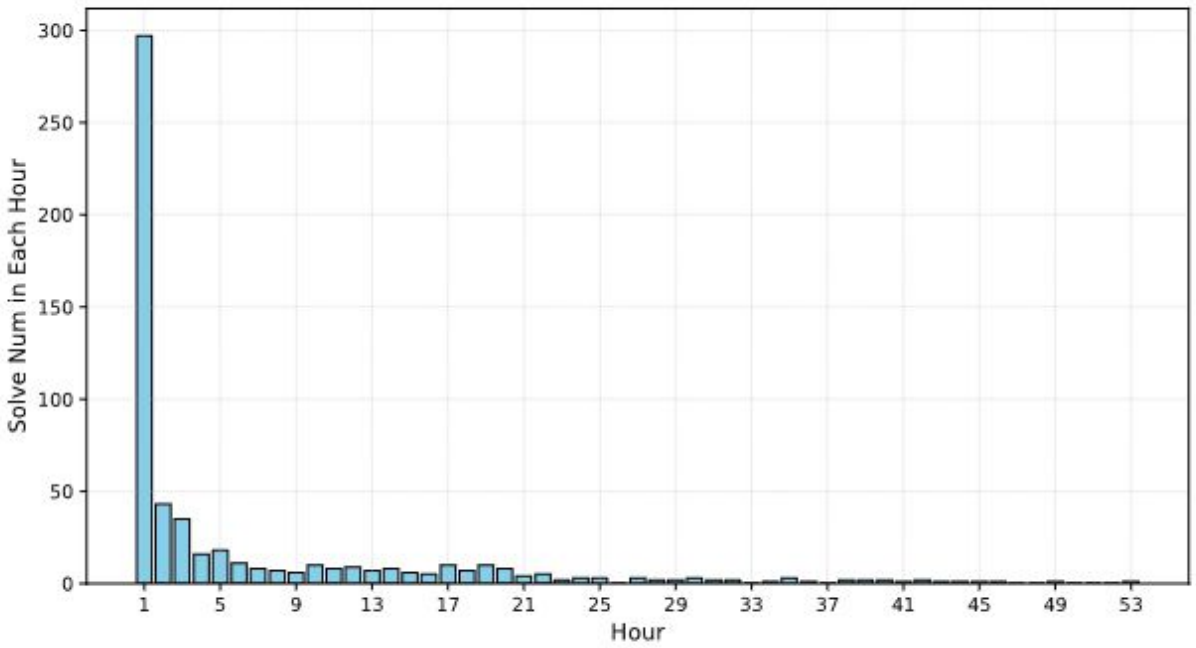
proves the easier misstatement

budget burn

spends the run on one dead end

Failure classes as discussed in ProofBench analyses (vals.ai) and agent logs

Some proofs live in a very long tail



PutnamBench

87.9%

580/660 · paper setting

FATE-H / X

80% / 33%

paper settings

- longest successful run: ≈ 53 hours;
- 11/12 Putnam 2025 problems within 9 hours.

After Seed-Prover 1.5 (2025), Fig. 6 · arXiv:2512.17260

Proof example (Hilbert)

```
theorem h_divides_amc12b_2002_p4 (n : ℕ) (h₀ : 0 < n) (h₁ : (1 / 2 + 1 / 3 + 1 / 7 + 1 / ↑n :
↪  ℚ).den = 1) (h_sum_is_frac : (1 / 2 + 1 / 3 + 1 / 7 + 1 / ↑n : ℚ) = (↑(41 * n + 42)) / ↑(42 *
↪  n)) : 42 * n ∣ 41 * n + 42 := by
  have h₂ : ((↑(41 * n + 42) : ℚ) / ↑(42 * n)).den = 1 := by
    have h₂₁ : ((↑(41 * n + 42) : ℚ) / ↑(42 * n)) = (1 / 2 + 1 / 3 + 1 / 7 + 1 / ↑n : ℚ) := by
      rw [h_sum_is_frac]
      <;> field_simp <;> ring_nf <;> norm_cast
      <;> field_simp <;> ring_nf <;> norm_cast
    rw [h₂₁]
    exact h₁

  have h₃ : (42 * n : ℤ) ∣ (41 * n + 42 : ℤ) := by
    have h₃₁ : (((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ)).den = 1 := by
      norm_cast at h₂ ⊢
      <;> simpa [div_eq_mul_inv] using h₂
    have h₃₂ : ∃ (k : ℤ), ((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ) = (k : ℚ) := by
      have h₃₃ : (((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ)).den = 1 := h₃₁
      have h₃₄ : ∃ (k : ℤ), ((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ) = (k : ℚ) := by
        have h₃₅ : ∃ (k : ℤ), ((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ) = (k : ℚ) := by
          have h₃₆ : (((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ)).den = 1 := h₃₁
          have h₃₇ : ∃ (k : ℤ), ((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ) = (k : ℚ) := by
            refine' ⟨(((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ)).num, _⟩
            have h₃₈ : (((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ)) = ((((41 * n + 42 : ℤ) : ℚ) / (42
            ↪  * n : ℤ)).num : ℚ) := by
              have h₃₉ : (((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ)).den = 1 := h₃₁
              have h₄₀ : (((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ)) = ((((41 * n + 42 : ℤ) : ℚ) /
              ↪  (42 * n : ℤ)).num : ℚ) := by
                rw [← Rat.num_div_den (((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ))]
                <;> field_simp [h₃₉]
                <;> norm_cast
                <;> simp_all [Rat.den_nz]
              exact h₄₀
            exact h₃₈
          exact h₃₇
        exact h₃₅
      exact h₃₄
    obtain ⟨k, h₃₃⟩ := h₃₂
    have h₃₄ : (42 * n : ℤ) ∣ (41 * n + 42 : ℤ) := by
      have h₃₅ : ((41 * n + 42 : ℤ) : ℚ) / (42 * n : ℤ) = (k : ℚ) := h₃₃
      have h₃₆ : (42 * n : ℤ) ≠ 0 := by
        have h₃₇ : (n : ℕ) > 0 := h₀
        have h₃₈ : (42 * n : ℤ) > 0 := by
          norm_cast
```

Kernel-accepted does not mean useful

one generated Putnam proof, to scale

1097 tokens (as found by search)

76 after ProofOptimizer

- ≈87% shorter on miniF2F, ≈57% on PutnamBench (49% on Seed-Prover's IMO 2025 proofs);
- ≈28% of shortened proofs check ≥1.5× faster;

ProofOptimizer (2025) arXiv:2510.15700

Kernel-accepted does not mean merge-ready

wrong abstraction
correct, but stated at the wrong generality

duplicate
already in the library, under another name

unnecessary imports
drags in half of Mathlib

unstable tactics
breaks on the next release

opaque proof
accepted but unreadable

poor name, no docs
unfindable, unusable

mismatch
fights the library's conventions

…and more
style, linting, review load

Kernel-accepted does not mean merge-ready

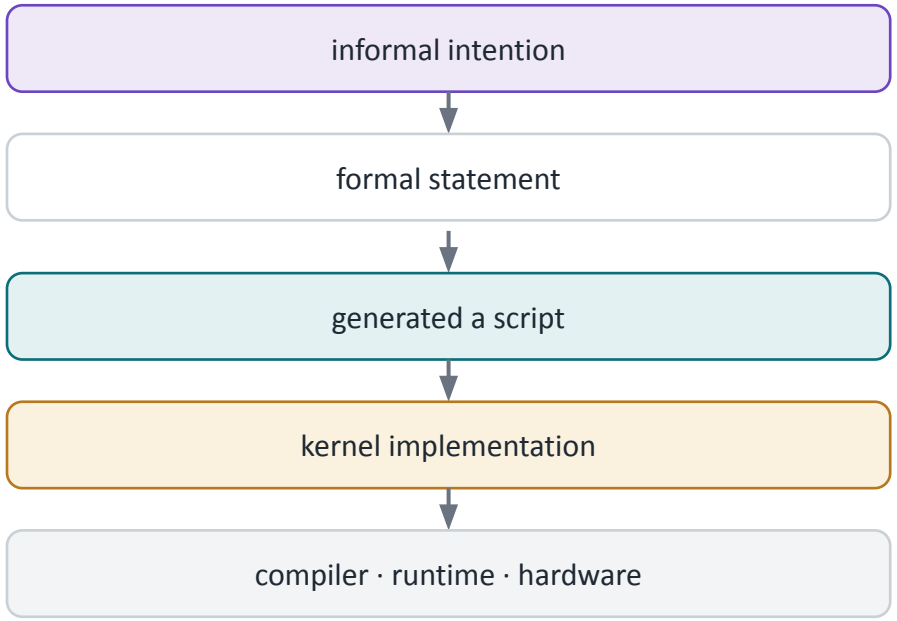
Kevin Buzzard 3:54

I think this is an important question; what to do with these thousands of lines of unmaintainable code which the AI wrote. math.inc are now in discussion with a number of groups, including groups working on things which I would like to see in mathlib, and I'm genuinely scared that if these systems start generating low-quality Lean code proving useful theorems and then writing the blog post and then moving on without any interest in getting the code tidied up and into mathlib then we are going to start accumulating technical debt at a slightly alarming rate.

The checker is not infallible software

2026 reality check

- **Lean 4.32.1 & 4.32.2 (July 2026):** kernel soundness bugs, metaprograms could prove False
- **Rocq (March 2026):** seven implementation paths to inconsistency found via AI-assisted testing



Take-away: from generation to engineering

1. A proof agent is a model inside a loop of tools, memory, search, feedback, and stopping decisions.

2. A verifier makes massive exploration useful, but does not make it cheap, unbiased, readable, or maintainable.

3. The frontier is moving from “prove an isolated theorem” to “contribute safely to a living formalization project.”