

LeanFlow

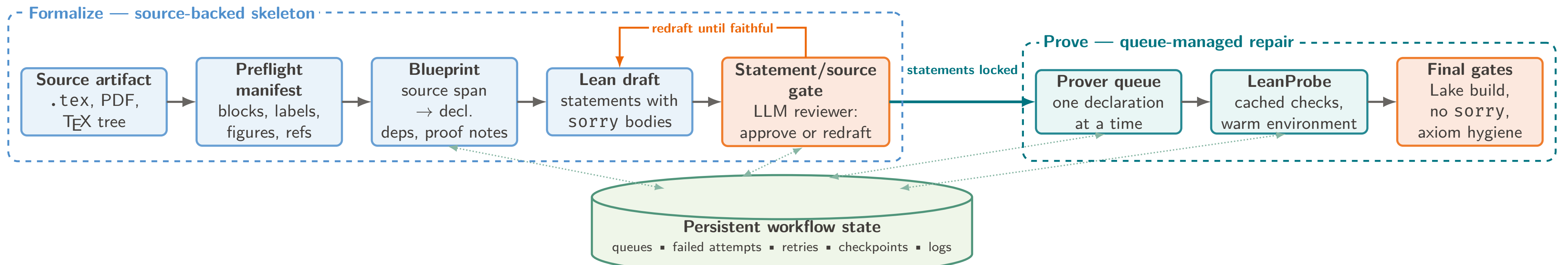
Workflow-Driven Lean Autoformalization

Lazar Milikić, Simon Guilloud, Khanh Nguyen, Viktor Kunčák
EPFL, Lausanne, Switzerland ■ ICML 2026 AI for Math Workshop

EPFL

From informal mathematical document to sorry-free Lean 4 project:
structured, source-gated statement translation and queue-based theorem proving.

From Document to Verified Lean Project



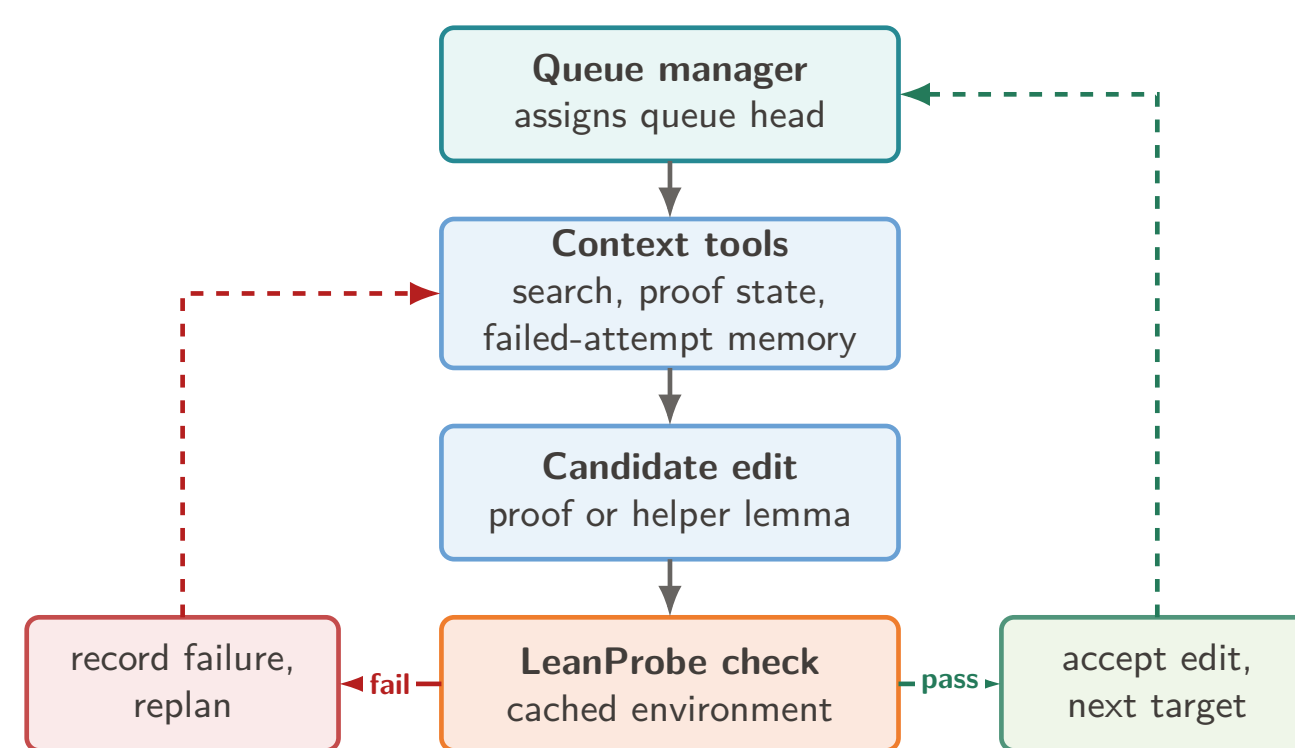
The formalizer drafts source-backed declarations and an autonomous reviewer gates statement faithfulness *before* proof search; the prover repairs one declaration at a time. All run state persists **outside the model context**, keeping runs resumable and auditable.

Why Naive Runs Break

- **Compiling $\neq$ faithful.** A well-typed theorem can formalize the *wrong* source claim — weakened conclusion, changed quantifiers, shifted domains. **LeanFlow's answer:** independent external reviews, in fresh contexts, gate each statement against its source — redrafting until faithful — before any proof search.
- **Transcripts forget.** Long runs lose failed-attempt memory after context compaction and repeat broken proof shapes.
- **Free agents wander.** Without routing, calls go to the wrong file, unstable dependencies, or already-failed paths.

Design rule. Separate statement faithfulness from proof completion: first approve *what each declaration means*, then let automated repair finish the proof under verifier control.

Inside the Prover Loop



The manager exposes **one active declaration** at a time, stores theorem-local failures and retry budgets, and advances only after external Lean verification accepts the edit. Statements stay frozen: repair may add helpers, never weaken the target. Checkpoints keep every run resumable and auditable.

Cached Lean Feedback

Cached same-file verifier (CLI ■ Python ■ MCP server): prepare the target's Lean environment once, then check each candidate in milliseconds. Failed candidates come back as annotated Lean:

```
theorem my_add_comm (x y : Nat) :  
  x + y = y + x := by  
  -- proof state: x y : Nat  
  --   ⊢ x + y = y + x  
  rfl  
  -- error: tactic 'rfl' failed
```

Latency per check:

full-file Lake 1.5–2.6 s
LeanProbe 0.03–0.05 s

Sequential file verification:

9–14× faster than
growing-prefix Lake checks.

Plugs into the repair loop of any Lean agent;
final acceptance stays with Lean/Lake.



epfl-lara/LeanProbe

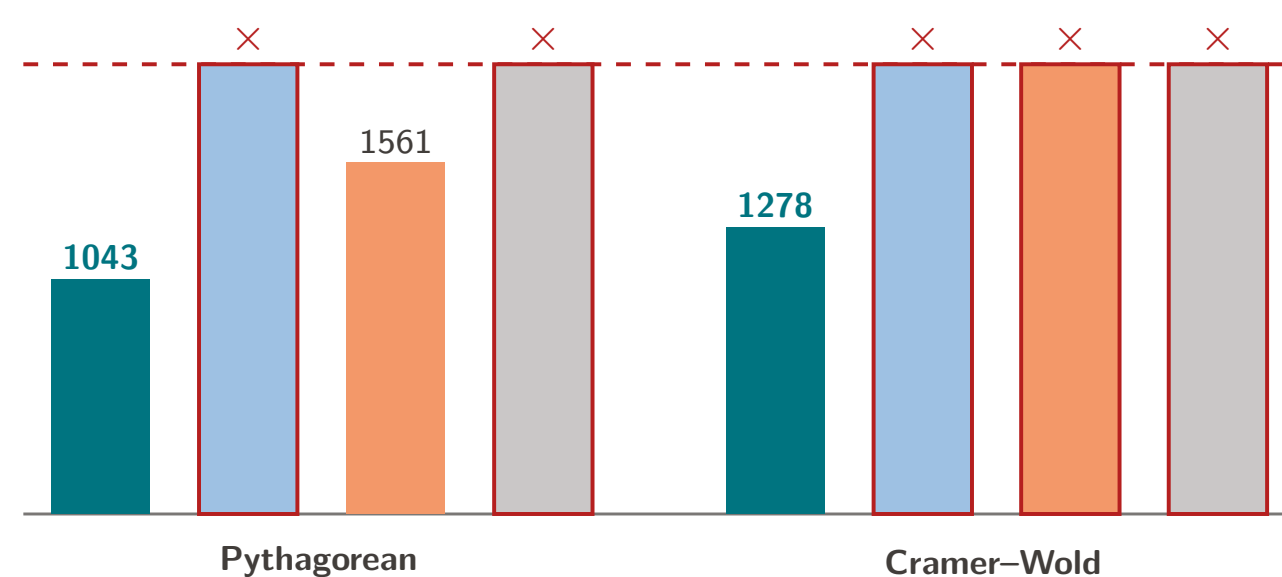
Do Workflows Matter? Ablations on Real Papers

Two papers — **Frisch–Vaserstein** (Pythagorean triples) and **Lyons–Zumbrun** (Cramer–Wold) — proved from the same reviewed skeleton under a 2000-call budget. Full LeanFlow uses both queue manager and tool surface; ablations remove queue and/or tools (+tools = context/proof-state tools + LeanProbe).

■ LeanFlow (queue + tools) ■ LeanFlow -queue ■ LeanFlow -tools ■ LeanFlow -queue -tools

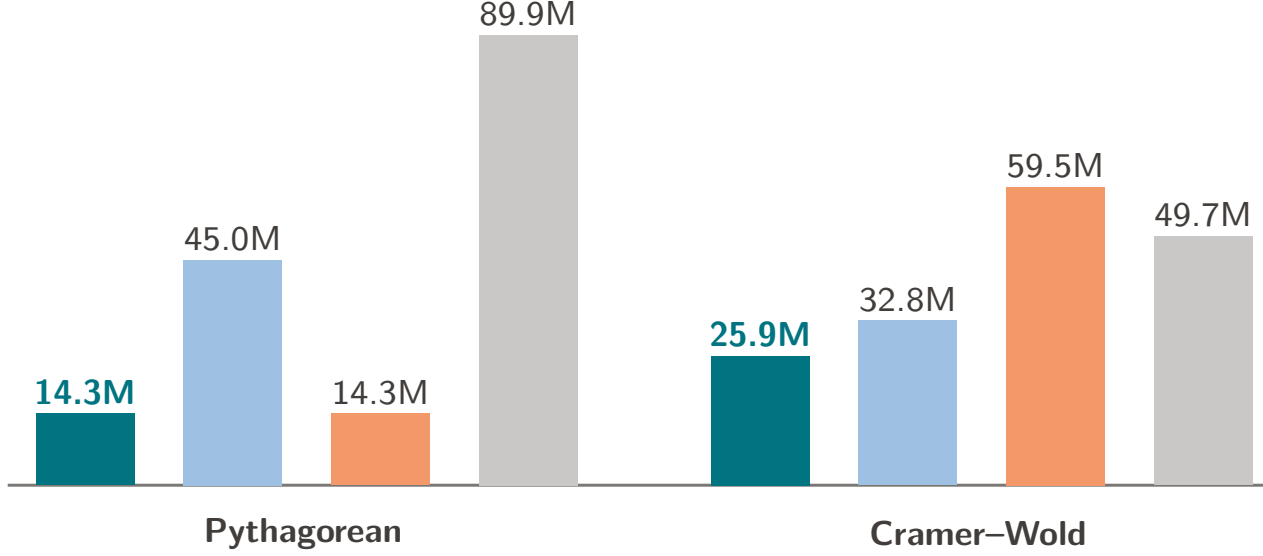
Kimi-K2.6: prover calls (lower is better)

2000-call cap = failure (×)



Queue control is **decisive**: every no-queue
Kimi-K2.6 run fails at the budget cap.

GPT-5.5: input tokens (all 8 variants succeed)



Every GPT-5.5 variant finishes: full LeanFlow is
cheapest or tied on input tokens.

Calibration (GPT-5.5): RLM25-PFR: 81.2% verified proofs, BEq+ 75.7% vs. 72.9% terminal-only, 13% fewer calls, 14% fewer input tokens ■ **AI4Math TCS:** 5 / 5 projects solved, LeanProbe in 37% of steps.

2nd place, ICML AI for Math Challenge 2026 Track 2: 6 / 7 topics solved.

Code & Artifacts

Each run ships as an **inspectable Lean project with a replayable workflow trace** — blueprints, logs, checkpoints, failed attempts — not just final answers. **Pythagorean:** 83 declarations incl. 34 theorems/lemmas ■ **Cramer–Wold:** 114 declarations incl. 101 theorems/lemmas — sorry-free, no unapproved axioms.

Both generated Lean projects:

github.com/epfl-lara/AutoformalizedProjects



github.com/epfl-lara/LeanFlow

runtime ■ workflow logs ■ benchmarks