# Conjecture Extraction for Proof Autoformalization

Simon Sorg  Wenda Li  Soumya Banerjee

University of Cambridge

# Outline

# Introduction to Diophantine Equations

## What is a Diophantine equation?

A *Diophantine equation* is a polynomial equation for which we seek *integer* (or sometimes rational) solutions. Typical form: $F(x_1, x_2, \ldots, x_n) = 0$ with $F \in \mathbb{Z}[x_1, \ldots, x_n]$.

## Common classes

- **Linear:** $ax + by = c$ (integer solutions; solvability tied to $\gcd(a, b)$).
- **Quadratic:** e.g. Pythagorean triples $x^2 + y^2 = z^2$.
- **Exponential / higher degree:** e.g. Fermat-type equations $x^n + y^n = z^n$.
- **Pell's equation:** $x^2 - Dy^2 = 1$ (infinitely many solutions for non-square $D$).

# Questions

## Typical questions

1. *Solvability*: does any integer solution exist?
2. *Parametrization*: can all solutions be described in closed form?
3. *Finiteness / infinitude*: are there finitely or infinitely many solutions?

# Fermat's Last Theorem: brief context

**Statement (Fermat, margin note ca. 1637)**

For integer exponent $n > 2$, the equation

$$x^n + y^n = z^n$$

has no nonzero integer solutions $x, y, z \in \mathbb{Z} \setminus \{0\}$.

**Historical highlights**

- Fermat claimed a marvelous proof in the margin of his copy of Diophantus (no proof was left).
- Many special cases were proven over centuries (e.g. $n = 4$ by Fermat himself using infinite descent; other small exponents by Euler, Sophie Germain, Kummer, etc.).
- 20th-century breakthroughs connected FLT to deep arithmetic geometry (elliptic curves and modular forms).

# Why it matters?

## Why it matters

- FLT is more than an isolated diophantine statement — its proof united areas of number theory, algebraic geometry, and the theory of modular forms.
- It illustrates how apparently elementary Diophantine problems can require powerful modern machinery.
- The techniques developed and refined (e.g. modularity, Galois representations) continue to drive current research in arithmetic geometry and Diophantine equations.

# Math in the 1600s

# Math in the age of AI?

# The Challenge of Formalization

- Formalization is costly and labor-intensive
- Requires significant expertise from a small group of specialists
- **Autoformalization**: Translating informal statements into formal versions using LLMs
- **ATP (Automated Theorem Proving)**: Proving formal conjectures

**Gap:** Few works focus on *proof autoformalization* (translating informal proofs into formal proofs)

# Current Limitations

## Draft, Sketch, and Prove (DSP)

- Originally designed for Isabelle
- Limited success in Lean 4 due to syntax adherence issues
- Incompatible with modern whole-proof generation models

**Need:** A proof autoformalization method for Lean 4 that works with whole-proof generation models

# Our Contribution: Conjecture Extraction

1. **Decompose** informal proof into individual lemmas (conjectures)
2. **Formalize** and prove each conjecture in isolation
3. **Reassemble** them to recover the original argument

## Key Advantages

- Compatible with whole-proof generation models
- Leverages repeated conjecture refinement
- Achieves **11.2% absolute improvement** over DSP on MiniF2F

# Introducing Lean: a quick taste

## What is Lean?

- Lean is a modern *interactive theorem prover / proof assistant* (Lean 4 is the current release).
- It is used to formalise mathematics and to verify programs and proofs with machine precision.
- Work mode: write definitions/theorems, then interactively build proofs using tactics or term-style proofs.

# Lean

```
1 import Mathlib.Logic.Basic -- basic facts in logic
2 -- theorems in Lean's mathematics library
3
4 -- Let P and Q be true-false statements
5 variable (P Q : Prop)
6
7 -- The following is a basic result in logic
8 example : ¬ (P ∧ Q) ↔ ¬ P ∨ ¬ Q := by
9   -- its proof is already in Lean's mathematics library
10  exact not_and_or
11
12 -- Here is another basic result in logic
13 example : ¬ (P ∨ Q) ↔ ¬ P ∧ ¬ Q := by
14  apply? -- we can search for the proof in the library
15  -- we can also replace `apply?` with its output
16
```

Figure: Lean code

# Conjecture Extraction Overview

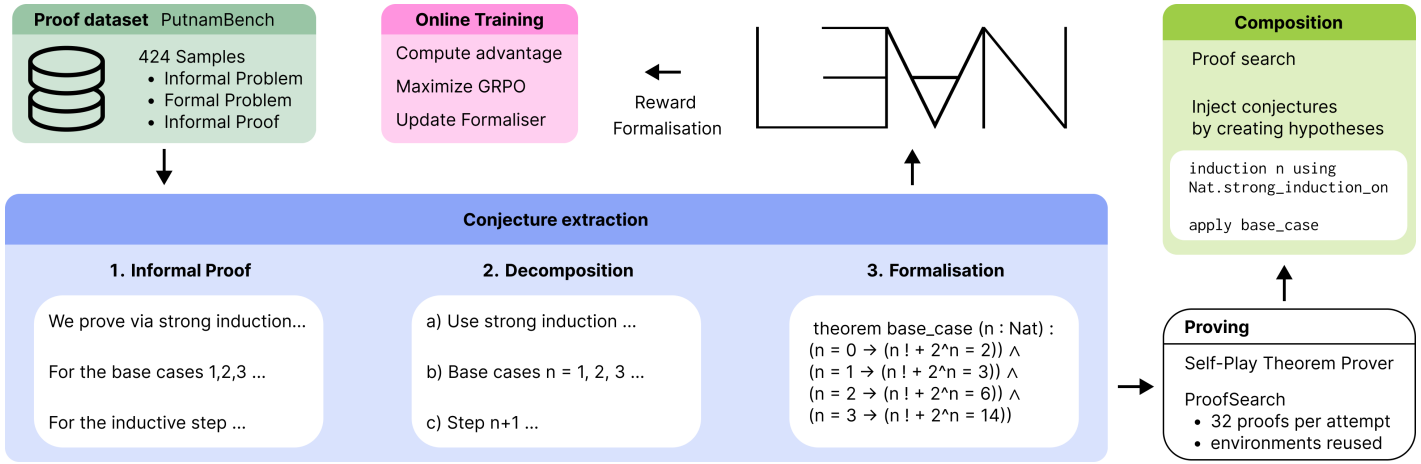



Figure: General overview of conjecture extraction pipeline

# Conjecture Extraction Process

**Input to LLM:**

- Informal problem statement
- Informal proof of the problem
- Formalized problem statement

**Output Format (4 parts):**

1. **Reasoning**: Chain of thought explanation
2. **Given**: Variable declarations and ranges
3. **Assumes**: Required hypotheses
4. **Shows**: Conclusion from variables and assumptions

# Autoformalization with Retry Mechanism

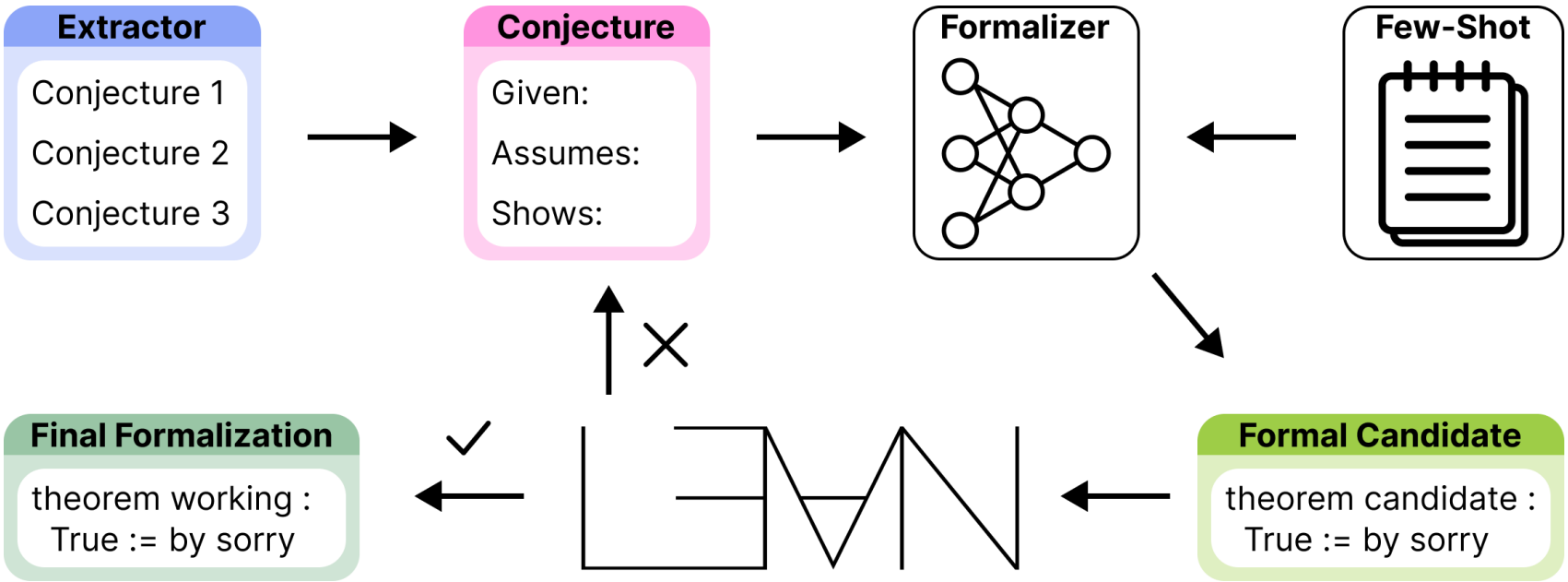



Figure: Iterative formalization with type-checking

- Generate formalization using one-shot in-context learning
- Type-check using Lean REPL
- Retry up to $F = 10$ times if errors occur

# Technical Challenges & Solutions

## 1. Naming Collisions

**Solution:** Use separate namespace per theorem (UUID-based)

## 2. Premise Selection

- Embed proof state using ReProver retrieval encoder
- Retrieve $k = 10$ nearest proven premises
- Use **hypothesis injection** to add premises efficiently

## 3. Hypothesis Rejection

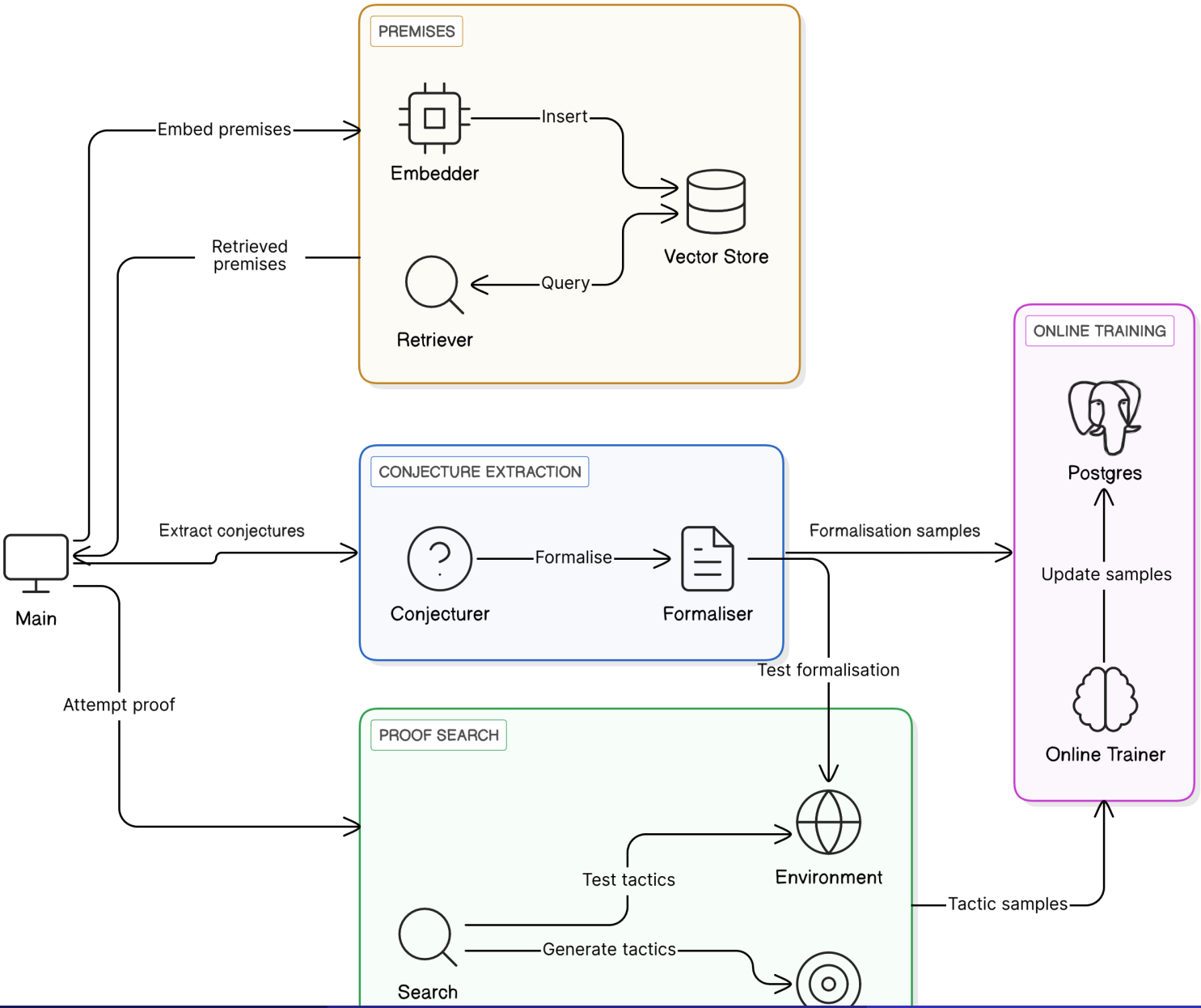
Replace conclusion with `False` and attempt proof to filter invalid assumptions

# Hypothesis Injection Strategy

Three possible approaches:

1. Explicitly add tactics (, )
   - – Enlarges search space significantly
2. Convert premises to  statements
   - – Results in very long proofs
3. **Transform premises into hypotheses**
   - \+ Efficient and provides sufficient information
   - \+ Chosen approach

# Open-Source Implementation



# Results: Comparison with DSP

Table: Proof rates on MiniF2F benchmark (pass@1)

| **Method** | **Validation** | **Test** |
|---|---|---|
| Aesop | 11.5% | 12.7% |
| Pantograph (DSP) | 12.7% | 14.7% |
| **Conjecture extraction** | **21.3%** | **25.9%** |

**11.2% absolute improvement** over DSP on MiniF2F test!

# Results: Proof-Step Generation (PutnamBench)

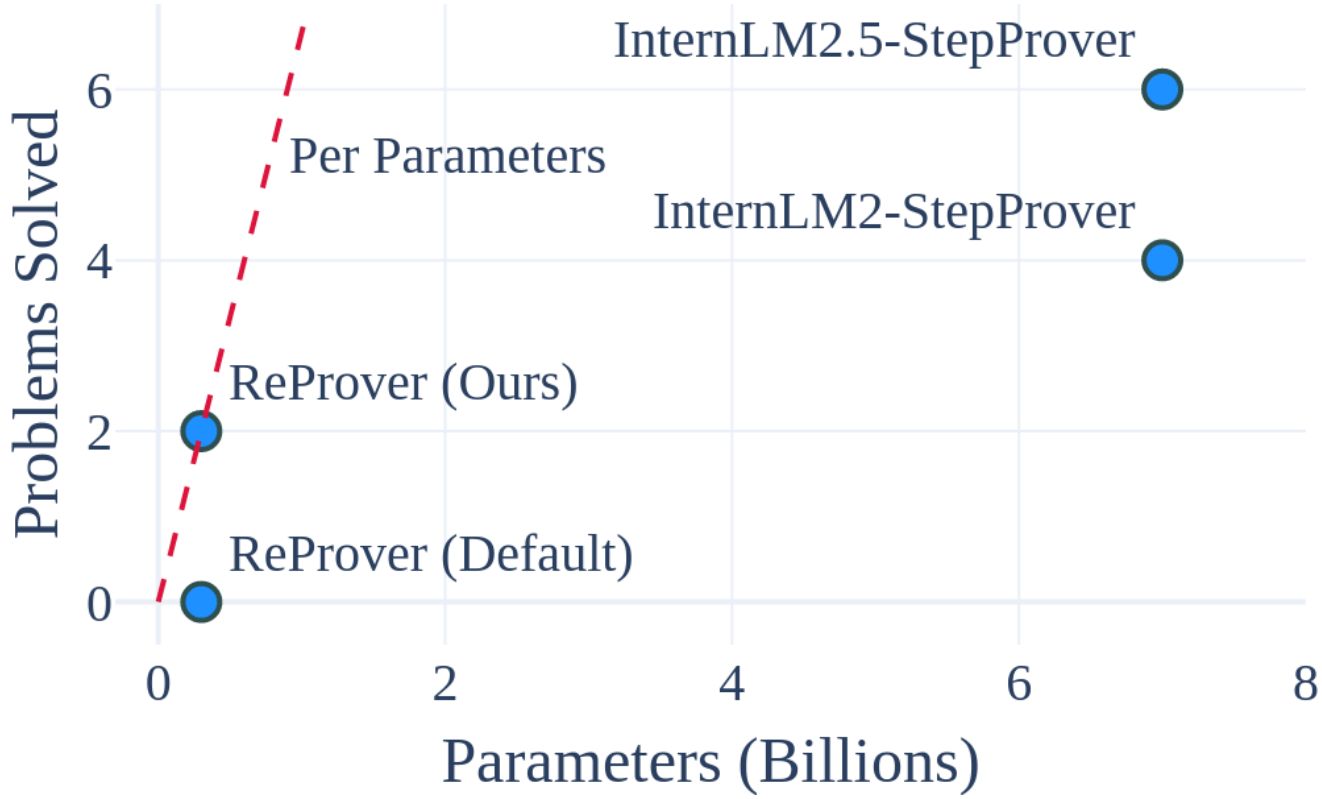



Figure: Proven Putnam problems vs. parameter counts

- ReProver + Conjecture Extraction: **2 problems solved**
- Baseline ReProver: 0 problems solved
- All outperforming models are **20× larger** (300M vs 7B parameters)

# Results: Whole-Proof Generation (MiniF2F)

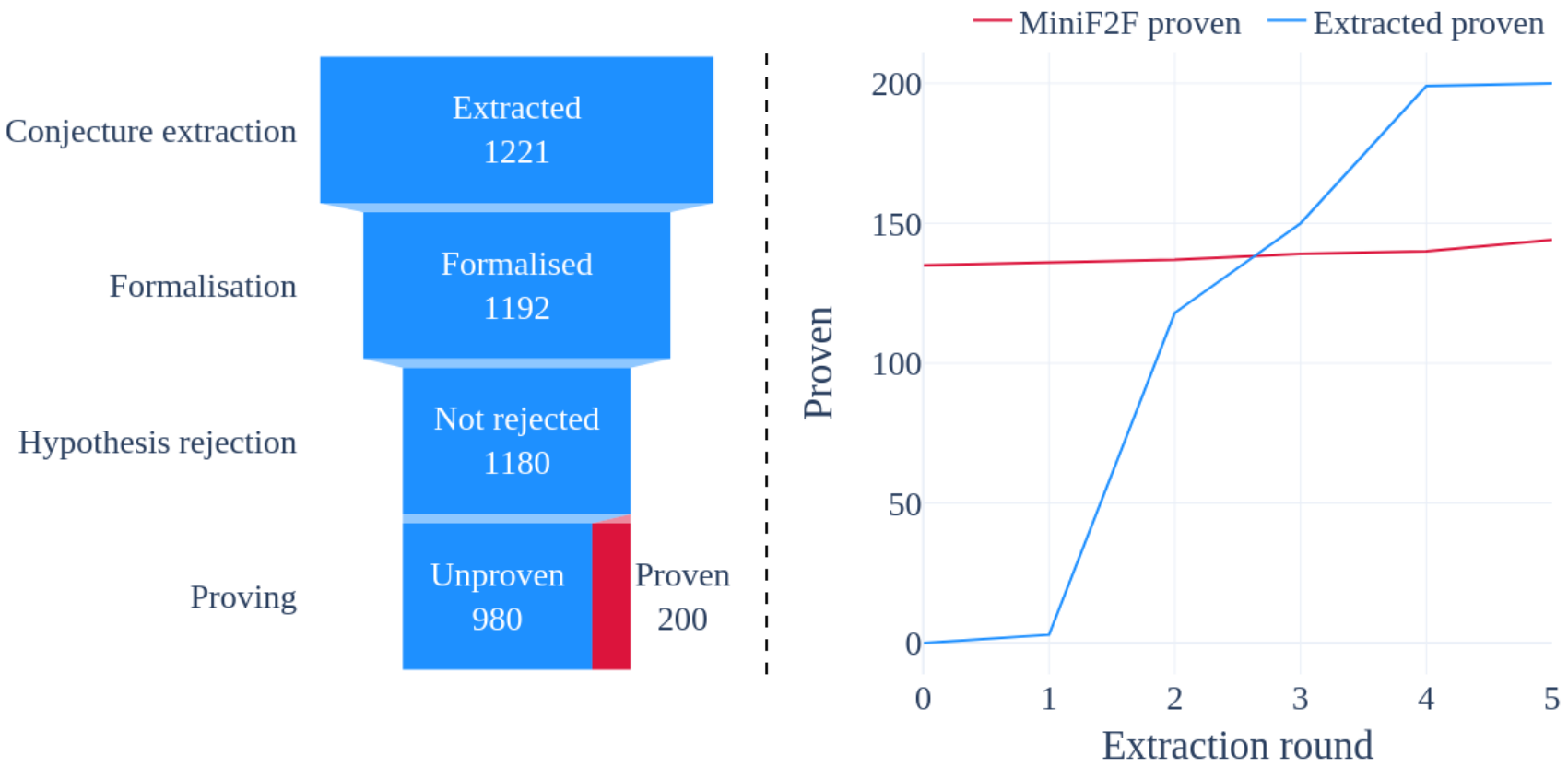



Figure: Left: Pipeline stages. Right: Iterative improvement

**Self-Play Theorem Prover results:**

- Without conjecture extraction: 135 problems (pass@32)
- With conjecture extraction: **144 problems** (pass@5×32)

# Power of Repeated Conjecture Extraction

**Statistics (5 rounds):**

- 1,192 conjectures extracted
- 200 successfully proven
- 12 rejected via hypothesis rejection

**Key Insight:**

Repeated extraction continuously boosts performance due to:

- Different proof step formulations
- Varying formalizations
- Progressive difficulty reduction

# Summary of Contributions

1. **Conjecture extraction**: Novel proof autoformalization for Lean 4
   - Compatible with whole-proof generation models
   - Does not rely on proof sketches
2. **Open-source implementation**
   - Combines ATP, autoformalization, conjecture generation
   - Online reinforcement learning support
3. **Strong experimental results**
   - 11.2% improvement over DSP on MiniF2F
   - First proof autoformalization for whole-proof generation

# Impact & Future Directions

## Broader Impact

- Bridges gap in proof autoformalization for Lean
- General framework scalable to diverse proof assistants
- Facilitates large-scale formalization efforts

## Future Work

- Extend to more complex mathematical domains
- Improve conjecture extraction accuracy
- Scale to research-level mathematics
- Integration with other proof assistants

# Math in the 1600s

# Math in the age of AI?

# Future work

- Can we reconstruct what Fermat may have been trying to prove?
- Can we come up with alternative proofs of FLT?

# Thank You

Questions?

**Code & Models Available:**

Open-source implementation released

`https://github.com/sorgfresser/conjectureextraction`

**Paper:**

`https://github.com/neelsoumya/paper_preprints/blob/master/Peru_MATH_AI_solve.pdf`

**Contact:**

`sb2333@cam.ac.uk`

`{tss52, wl302}@cam.ac.uk`