

Deep Learning for Mathematical Reasoning

Christian Szegedy

4/16/2024

What is Mathematical Reasoning

- “Informal” reasoning in natural language
 - Solving a natural language question by giving a natural language answer
 - Examples:
 - Elementary school math problems
 - Simple calculations
 - Word problems
 - High-school-level problems:
 - Simple calculations
 - Word problems
 - Multiple-choice questions
 - Simple Proofs
 - Research level mathematics:
 - Long proofs
 - Diagrams
 - Convincing a fellow mathematician in a long conversation

Reasoning in a fully formalized Language

- Given a fully formally specified problem in some formal theorem prover
- Create a proof
 - Syntactically correct
 - Akin to programming

```
import data.nat
open nat

definition even (a : nat) := ∃ b, a = 2*b
theorem EvenPlusEven {a b : nat} (H1 : even a) (H2 : even b) : even (a + b) :=
exists_elim H1 (fun (w1 : nat) (Hw1 : a = 2*w1),
exists_elim H2 (fun (w2 : nat) (Hw2 : b = 2*w2),
exists_intro (w1 + w2)
  (calc a + b = 2*w1 + b      : {Hw1}
... = 2*w1 + 2*w2      : {Hw2}
... = 2*(w1 + w2)      : eq.symm !mul.distr_left)))
```

Lean 4.0
Proof
(Code)

Formal vs Informal

Formal Reasoning:

- 100% guaranteed, correctness
- Fully specified
- Computer checkable
- Suitable for verifying software

Informal Reasoning:

- Semantics of correctness is fuzzy
- Problem formulation might be fuzzy
- Needs humans to verify
- Cumbersome/not very precise for verifying software.

Formal vs Informal

Formal Reasoning:

- 100% guaranteed, correctness
- Fully specified
- Computer checkable
- Suitable for verifying software

Very few well-accepted
AI-benchmarks

Informal Reasoning:

- Semantics of correctness is fuzzy
- Problem formulation might be fuzzy
- Needs humans to verify
- Cumbersome/not very precise for verifying software.

Several well-studied
AI-benchmarks: MATH, GSM8K, ...

Formal vs Informal

Formal Reasoning:

- 100% guaranteed, correctness
- Fully specified
- Computer checkable
- Suitable for verifying software

Very little unsupervised data available.

Informal Reasoning:

- Semantics of correctness is fuzzy
- Problem formulation might be fuzzy
- Needs humans to verify
- Cumbersome/not very precise for verifying software.

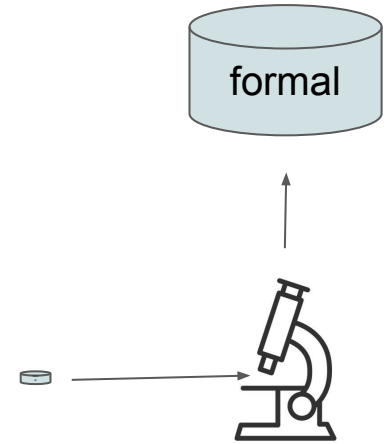
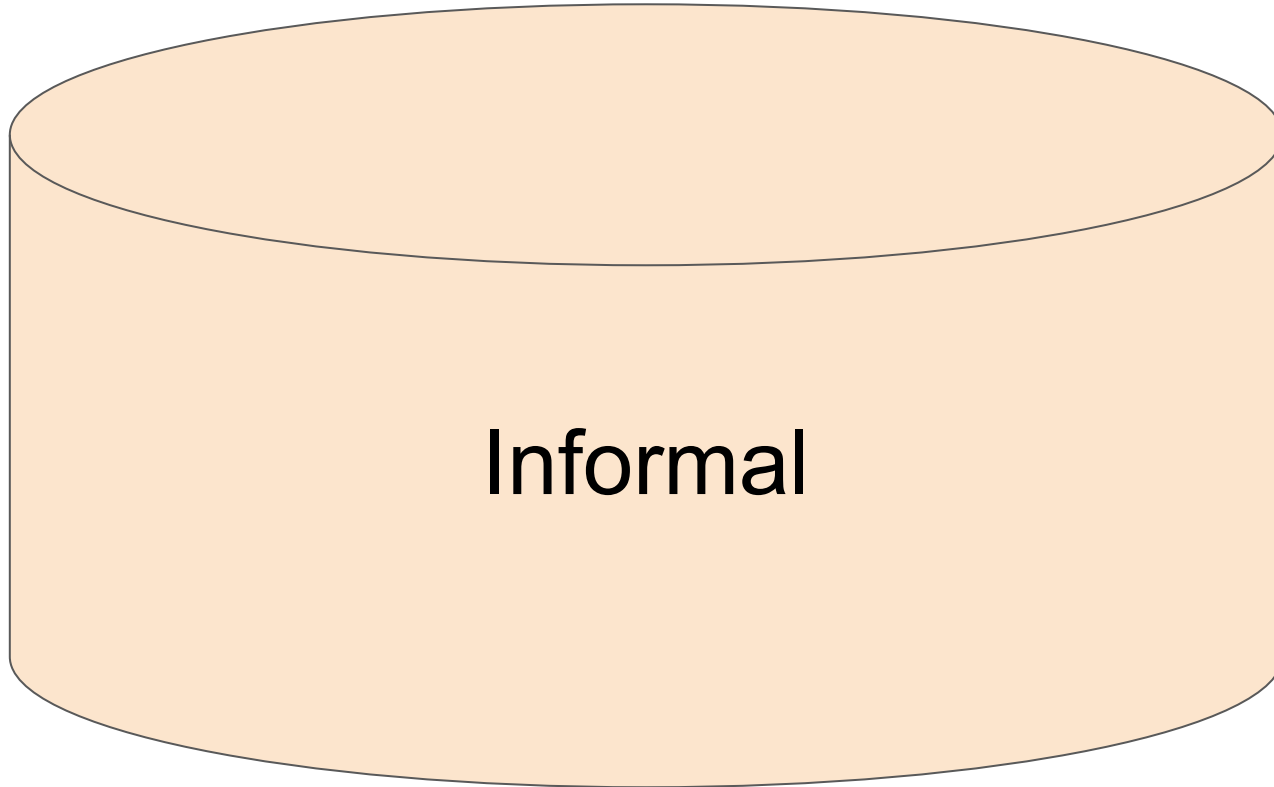
Large amount of unsupervised data: papers/books web pages.

Issues with formal proof checking

Historically: many provers, with different foundations:

- Mizar First order logic + axiom schemas
- HOL4 HOL / Higher Order
- Isabelle/HOL HOL
- HOL-Light HOL
- Coq HOL / Calculus of Constructions
- Metamath Own logic
- Lean HOL / Calculus of Constructions

Data: Formal vs Informal



What is Mathematics?

- Calculating some elementary arithmetic calculations?
- Solving hard problems such as the $P \neq NP$, Riemann Hypothesis or Collatz conjecture?

What is Mathematics?

- Calculating some elementary arithmetic calculations?
- Solving hard problems such as the $P \neq NP$, Riemann Hypothesis or Collatz conjecture?

Alternatively:

Mathematics is everything that can be done fully formally inside a computer without real world interaction and with efficiently, independently verifiable outcome.

Examples of “mathematics”

- What is the best move in a certain chess position?
- Is my hand likely to win given fully rational agents at given poker table and history of bets.
- What are the most general type annotations make this program type-check?
- Does this program ever throw an exception?
- Predict tomorrow's weather given today's weather data
- Compute the mass of the proton using QED
- What is the highest learning rate for this neural network to train on a given input distribution?

Examples of non-mathematics

- Self-driving
- Figuring out the laws of the universe
- Categorize today's news on the internet
- Predict what content will a certain user enjoy
- Is there life on Europa?
- Robotics
- Curing cancer

These problems require:

- Real world interaction/information acquisition
- Non-specified rules

Correctness is not verifiable formally (due to lack of rules).

Is Mathematics = Computation?

What is computation?

- Local changes
- Using well-defined, finite rules
- On arbitrary sized objects.

What is Computation?

Most common models of computation:

- Turing Machine
- Lambda Calculus

What is Computation?

Most common models of computation:

- Turing Machine
 - **Ugly:**
 - Relatively complex and obscure
 - Uses actual infinity for modeling potential infinity
 - Overly complex to model with actual code
 - Cumbersome to formalize precisely
- Lambda Calculus
 - Assumes a lot of ad hoc structure

What is computation

- Local changes
- Using well-defined rules
- On arbitrary sized objects.

Circular Turing Machine:

Finite state machines on the prefix/postfix pairs of strings

Simple model of computation

Let A be a finite alphabet and f and g two maps from $A \cup \{\emptyset\}$ to $A^2 \cup A \cup \{\emptyset, HALT\}$,

(\emptyset denotes the empty string), then a “circular” Turing machine is a deterministic process that iterates the following function from A^* to A^* :

$$F(x_1 \cdots x_n) = f(x_1, x_n)1x_2 \cdots x_{n-1}g(x_1, x_n).$$

Python implementation (iterate “step” until a HALT state is produced by f or g).

```
def step(f, g, w):  
    x = w[:1], w[1:][-1:]  
    return f[x] + w[1:-1] + g[x]
```

The formalization of formalization

Let's imagine a proof checking program (verifier). "p is a proof of s" iff:

$$V(p) = s$$

We also want the verifier to run in polynomial (close-to-linear) time.

The goal of theorem proving is to find such a "proof" p.

We also require that there is another (easy to compute) program "negation" N, such that $\{N(s), s\} \not\subseteq \text{Im}(V)$.

What is semantics?

We request a self-modeling ability:

- We want an Embedding algorithm, encoding of the process of computation into such a language:

If $T = TM(f, g)$ is a (circulant) Turing machine, then we require to have a computable “embedding E ”

$E(f, g, x, y)$ in $\text{Im}(V)$ iff $T(x)$ stops with output y , and there should be a polynomial time algorithm (in the execution time of $T(x)$) to compute $V(p) = E(f, g, x, y)$.

Also we require another computable embedding $F(f, g, x)$ such that if $F(f, g, x)$ in $\text{Im}(V)$, then T never halts.

What is mathematics

Mathematics is everything that can be described fully formally:

- Uses computation only
 - Using a finite set of locally applied rules
 - On arbitrarily long inputs
- Formal system
 - Efficiently verifiable
 - With self-referential semantics
 - Can represent computation by any Turing machine efficiently
 - Can prove the non-halting property of **some** Turing machines

The Vision of AI for mathematics

- Creating an artificial mathematician that (potentially) exceeds the problem-solving capability of any human mathematician.

Formal or Informal?

Formal vs Informal

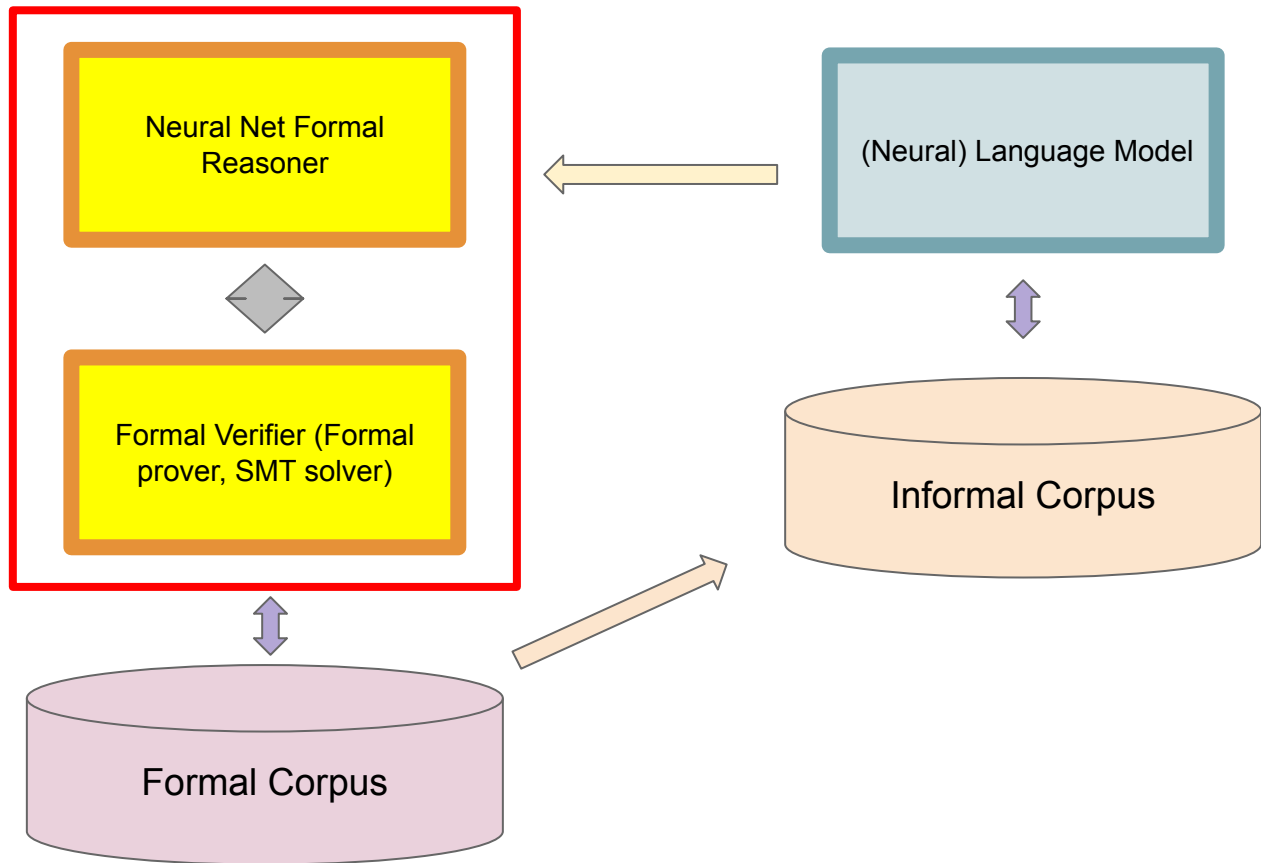
Formal only:

- If it is fully formal, we have much less training data to bootstrap from
- Hard to create a curriculum as even most theorems are not formalized.
 - Self play?
 - Even if we figure out what is interestingness, how would we communicate with a fully-formal system that uses a completely alien “code base”

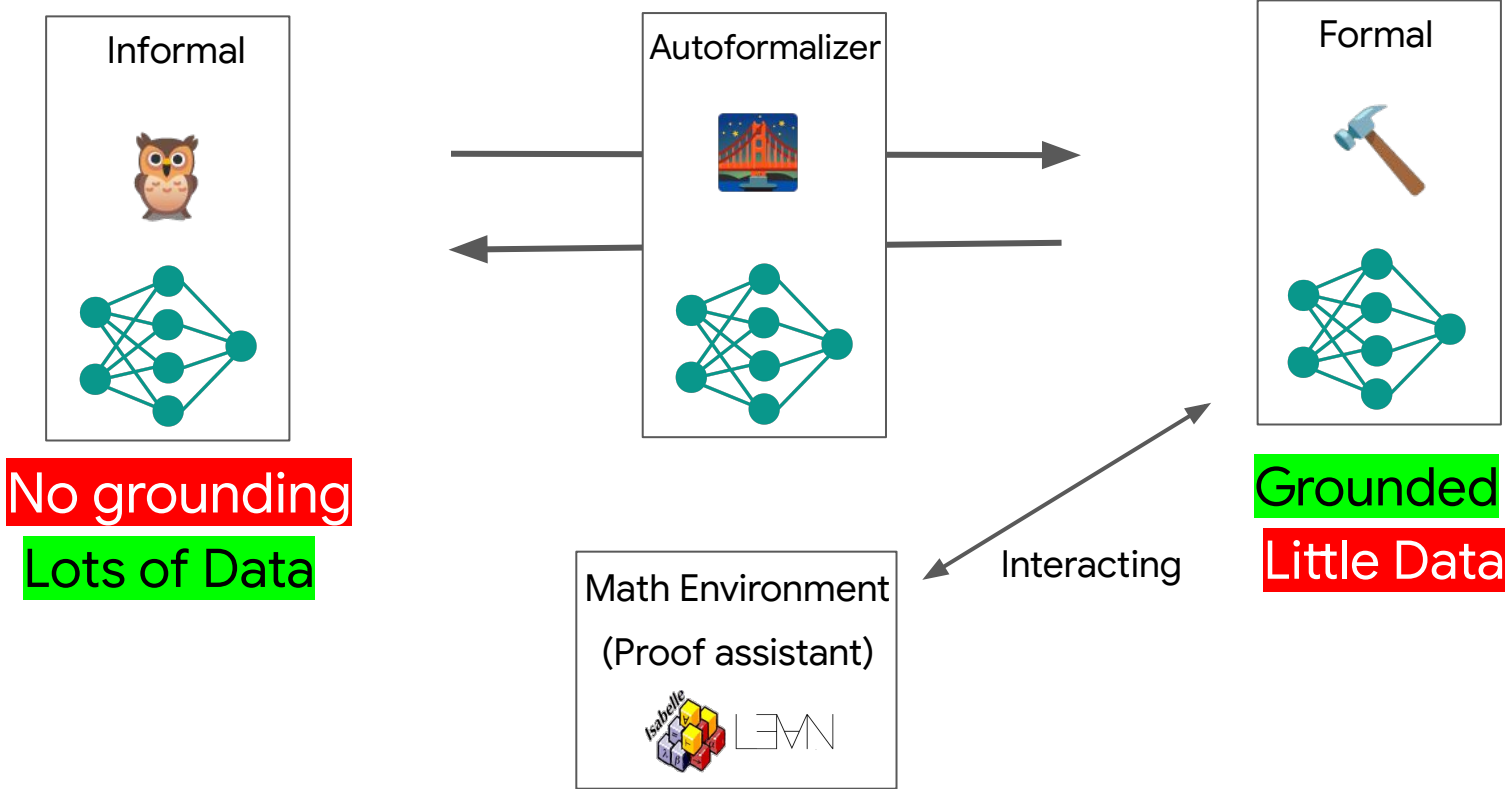
Informal only:

- How do we trust whether the system produces correct answers?
- How do we bootstrap its reasoning and evaluate its progress?

The Vision of Autoformalization



Autoformalization



Autoformalization

It's not easy...

- Task itself is extremely difficult.
 - Many missing information from informal proofs.
- We lack datasets to train machine learning models.
 - Very little amount of aligned pairs of “informal \leftrightarrow formal”.

Hmmm, what if LLM is so powerful...



Autoformalization with LLMs

Autoformalization with LLMs (NeurIPS 2022):

W. Albert Jiang, Wenda Li, Charles Staats, Markus Rabe, Mateja Jamnik, Christian Szegedy.

- First proof of concept of autoformalization with LLMs.
- Autoformalization of **statements**

Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs:

W. Albert Jiang*, Sean Welleck*, Jin Zhou*, Timothy Lacroix, Gary Liu, Wenda Li, Mateja Jamnik, Guillaume Lample.

- Autoformalization of **proofs**

Case studies: Main takeaways

1. LLMs can formalize despite little data in the pre-training corpus.
2. Capable of formalization completely out of domain problems!
3. We can teach the model to do better by showing relevant examples in the prompt.

Autoformalization of problem statements

Start with a 2-shot examples prompt:

Natural language version: "Let $z = \frac{1+i}{\sqrt{2}}$, find $(\sum_{i=1}^1 2(z^{i^2})) \cdot (\sum_{i=1}^1 2(\frac{1}{z^{i^2}}))$. The final answer is 36." Translate the natural language version to an Isabelle version:

```
theorem
  fixes z::complex
  assumes h0: "z = (Complex (1/sqrt 2) (1/sqrt 2))"
  shows "(\ $\sum_{k=1}^{12} (z^{k^2})$ )
    * ( $\sum_{k=1}^{12} 1/(z^{k^2})$ )=36"
```

Natural language version: "Determine the value of ab if $\log_8 a + \log_4 b^2 = 5$ and $\log_8 b + \log_4 a^2 = 7$. The final answer is 512". Translate the natural language version to an Isabelle version:

```
theorem
  fixes a b ::real
  assumes "(ln a) / (ln 8) + (ln (b^2)) / (ln 4) = 5"
    "(ln b) / (ln 8) + (ln (a^2)) / (ln 4) = 7"
  shows "a * b = 512"
```

Case Study 1

Wow!

- “there is no function f...” formalized to “assuming there is such a function proving “False””.
- The model understands what it means by the phrase “to itself”.

Case Study 1 Question:

"Prove that there is no function f from the set of non-negative integers into itself such that $f(f(n)) = n + 1987$ for every n ."

Codex Output:

```
theorem
```

```
  fixes f :: "nat \ $\rightarrow$  nat"
```

```
  assumes "\forall n. f (f n) = n + 1987"
```

```
  shows False
```

Case Study 2

Wow!

- Grade school math problem! Such a statement is completely out-of-distribution!
- The model did it perfectly.

Case Study 2 Question:

Natural Language version: *"When all the girls at Madeline's school line up in rows of eight, there are seven left over. If instead they line up in rows of four, how many are left over? The final answer is 3."* Translate the natural language version to an Isabelle version:

PaLM output:

```
theorem
```

```
  fixes n :: nat
```

```
  assumes "n mod 8 = 7"
```

```
  shows "n mod 4 = 3"
```


Case Study 3

A failure case:

- It fails to formalize the concept of “linear function”. It made up a name: linear f.

Case Study 3 Question:

Natural language version: *"Let f be a linear function for which $f(6) - f(2) = 12$. What is $f(12) - f(2)$? The final answer is 30."* Translate the natural language version to an Isabelle version:

Codex output:

theorem

```
fixes f :: "real \ $\rightarrow$  real"
```

```
assumes "linear f"
```

```
"f 6 - f 2 = 12"
```

```
shows "f 12 - f 2 = 30"
```

Teaching the model “what is a line” in the prompt

Add this new example to the prompt that explains a “line”.

Natural language version: "What is the x -coordinate for the x -intercept of the line containing the points $(7, 4)$ and $(6, 3)$? The final answer is 3". Translate the natural language version to an Isabelle version:

```
theorem
  fixes a b :: real
  and f :: "real \ $\rightarrow$  real"
  assumes h0 : "\ $\wedge$ x. f x = a * x + b"
  and h1 : "f 7 = 4"
  and h2 : "f 6 = 3"
  shows "f 3 = 0"
```



The model can now formalize a “linear function”!

Question:

Natural language version: "Let f be a linear function for which $f(6) - f(2) = 12$. What is $f(12) - f(2)$? The final answer is 30."

Translate the natural language version to an Isabelle version:

Codex Output:

```
theorem
  fixes a b :: real
  and f :: "real \ $\rightarrow$  real"
  assumes h0 : "\ $\wedge$ x. f x = a * x + b"
  and h1 : "f 6 - f 2 = 12"
  shows "f 12 - f 2 = 30"
```

More evals: Human Evals

- Codex is capable of translating 38 out of 150 MATH problems perfectly.
- Majority of the failures: **misalignment of definitions!**
 - “the greatest possible value” vs. “Greatest/Max”
 - “Factorial of n” vs. “fact n”

Failure cases \ Subjects	algebra	number_theory	inter_alg
Perfect translation	13	17	8
Incomplete/ill-formed/unclear prompt	9	3	14
Fail to align definitions or concepts	10	18	18
Inconsistent/missing assumption	8	9	9
Syntactical/type error	7	2	11
Missing definition in Isabelle	0	12	3
Wrong application of functions	6	13	16
Other	6	2	1

DSP framework

Informal statement

Statement

If $\gcd(n, 4) = 1$ and
 $\text{lcm}(n, 4) = 28$,
show that n is 7.



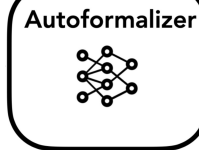
Draft

Informal proof

We know that $\gcd(a, b) \cdot \text{lcm}(a, b) = ab$,
hence $1 \cdot 28 = n \cdot 4$.

Then $n = 1 \cdot 28/4 = 7$,

completing the proof. ■



Sketch

Formal sketch

```
have c1: "1*28 = n*4"  
using assms
```

```
<proof>
```

```
then have c2: "n = 1*28/4"
```

```
<proof>
```

```
then show ?thesis
```

```
<proof>
```

Formal statement

```
theorem mathd_numbertheory_100:  
  fixes n :: nat  
  assumes "gcd n 4 = 1"  
    and "lcm n 4 = 28"  
  shows "n = 7"
```

Verified formal proof

```
have c1: "1*28 = n*4"  
using assms
```

```
by (smt (z3) prod_gcd_lcm_nat)
```

```
then have c2: "n = 1*28/4"
```

```
by auto
```

```
then show ?thesis
```

```
by auto
```

Prove

Off-the-shelf
Prover



Example – Minerva solves an IMO Problem

Statement: Prove that the fraction $\frac{21n+4}{14n+3}$ is irreducible for every natural number n .

Informal Proof (Minerva 62B):

We must show that $\gcd(21n + 4, 14n + 3) = 1$.

The Euclidean algorithm gives

$$21n + 4 = 1 \cdot (14n + 3) + 7n + 1$$

$$14n + 3 = 2 \cdot (7n + 1) + 1.$$

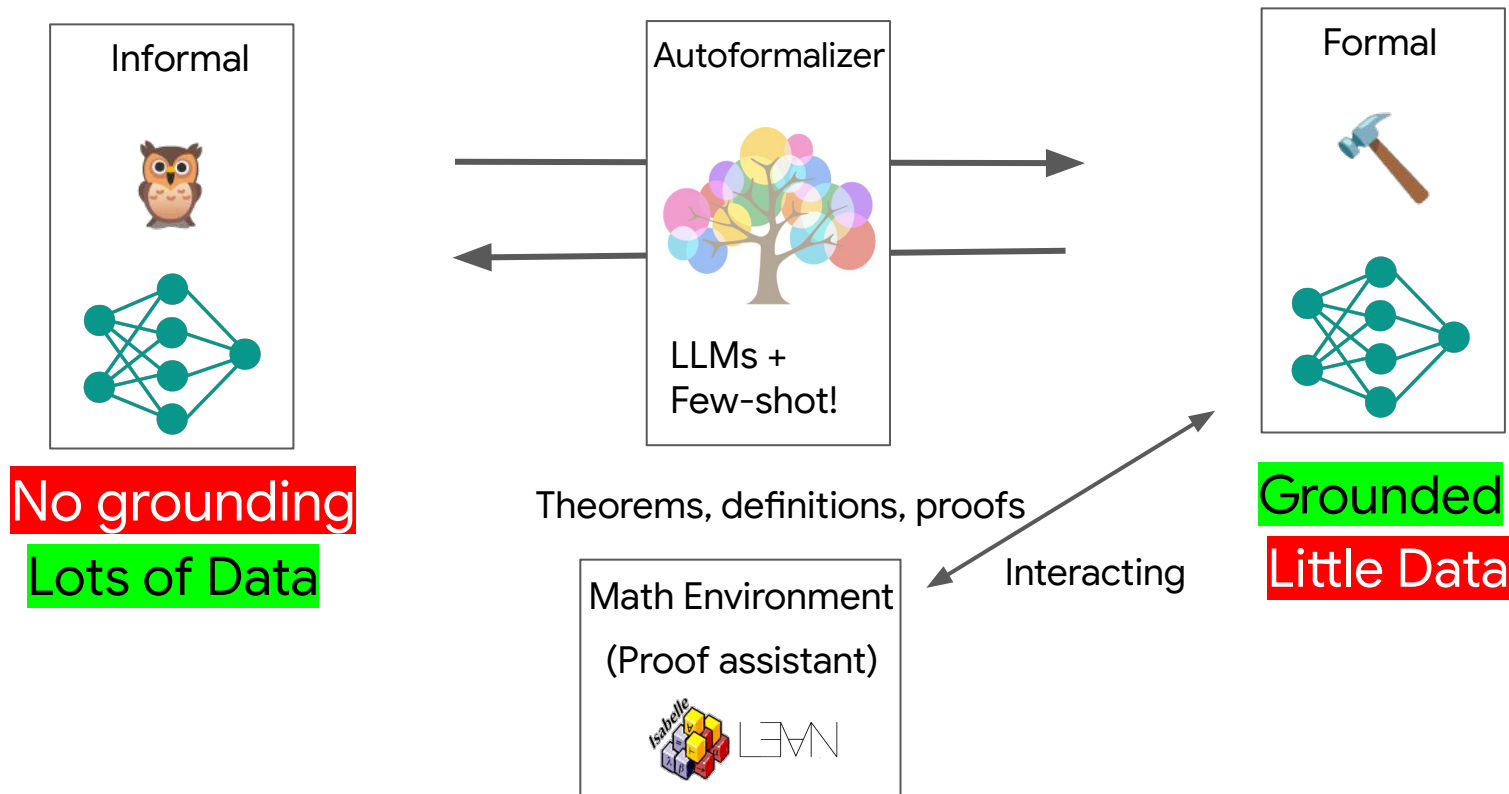
Since $\gcd(7n + 1, 1) = 1$, we have $\gcd(21n + 4, 14n + 3) = 1$.

Formal Proof:

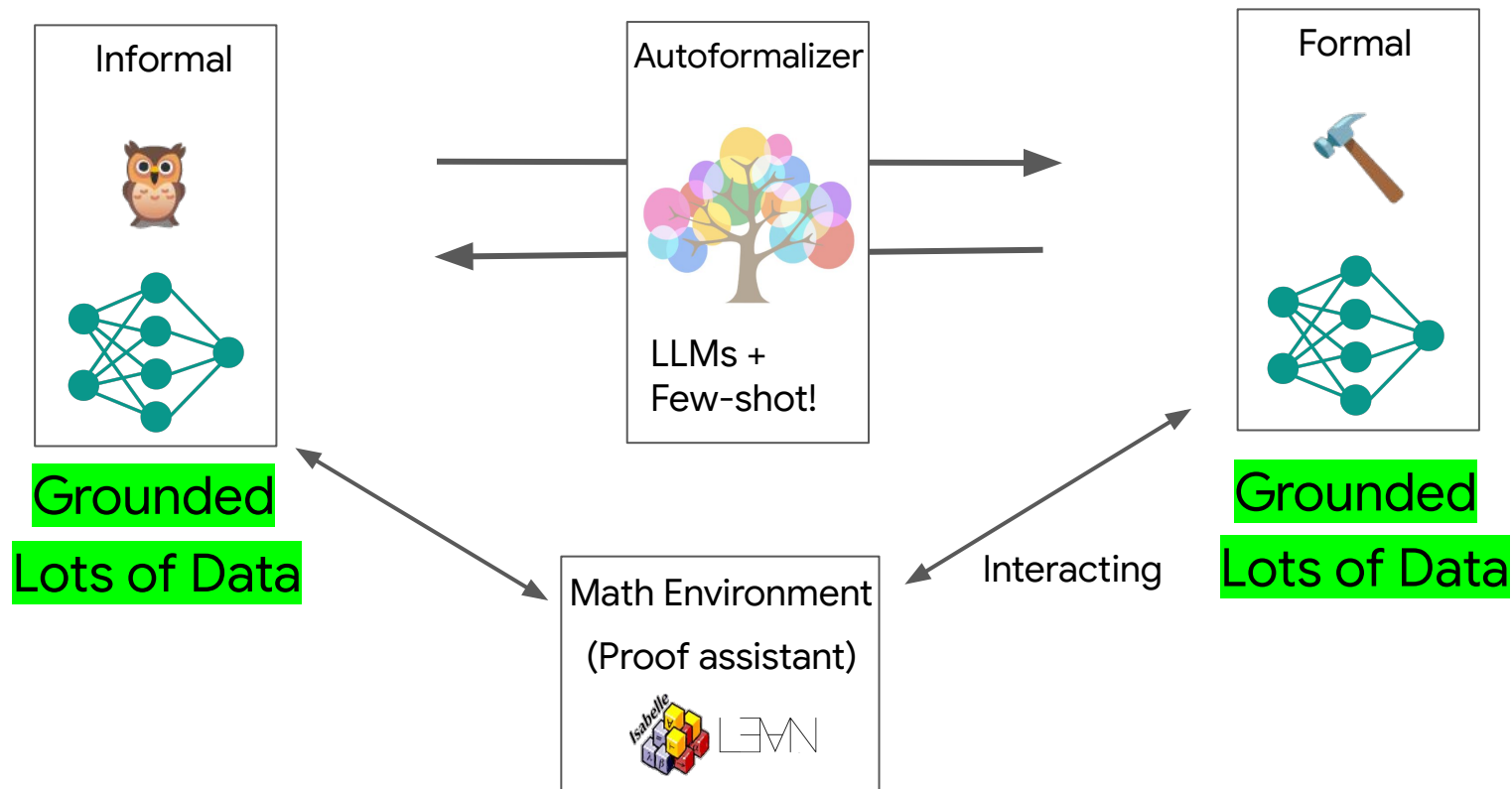
```
theorem imo_1959_p1:
  fixes n :: nat
  shows "gcd (21*n + 4) (14*n + 3) = 1"
proof -
  (* The Euclidean algorithm gives
  21n+4=1*cdot(14n+3)+7n+1
  14n+3=2*cdot(7n+1)+1. *)
  have c0: "21*n + 4 = 1*(14*n + 3) + 7*n + 1"
    <ATP> by auto </ATP>
  have c1: "14*n + 3 = 2*(7*n + 1) + 1" using c0
    <ATP> by auto </ATP>

  (* Since \gcd(7n+1,1)=1, we have \gcd(21n+4,14n+3)=1. *)
  then have "gcd (7*n + 1) 1 = 1"
    using c1
    <ATP> by auto </ATP>
  then have "gcd (21*n + 4) (14*n + 3) = 1"
    using c1
    <ATP> by (smt (z3) BitM_plus_one ab_semigroup_add_class.add_ac(1)
    add.assoc c0 gcd.commute gcd_add2 gcd_add_mult mult_numeral_1
    numeral_One numeral_eq_Suc numerals(1) semiring_norm(3)) </ATP>
  then show ?thesis
    using c1
    <ATP> by blast </ATP>
qed
```

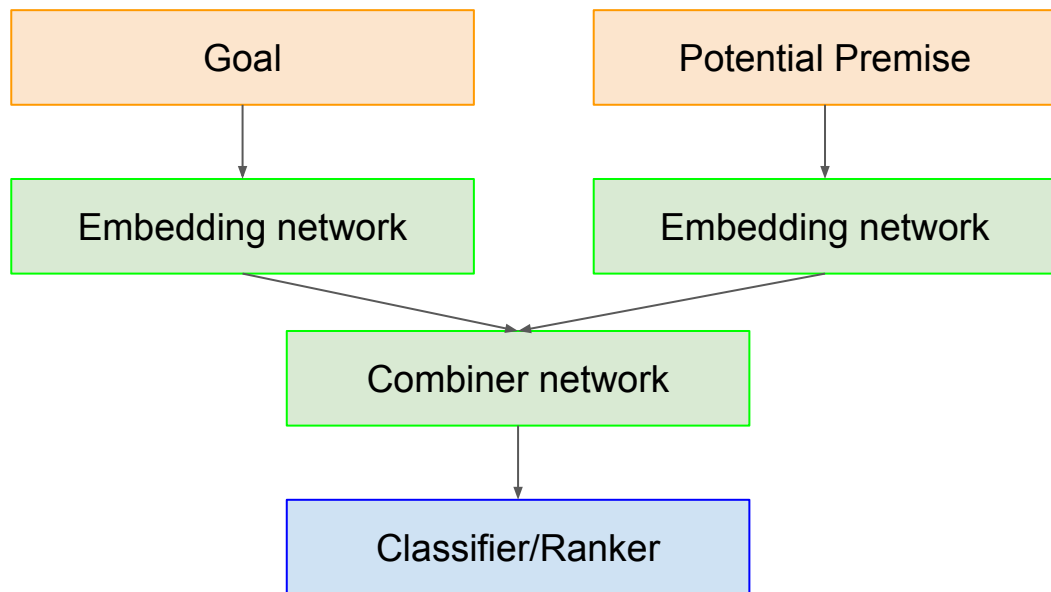
Autoformalization with LLMs



Future of mathematics



Premise Selection from Knowledge Base



Embedding Network:

- **Convolutional** network
- Recurrent **LSTM** network
- WaveNet style network
- Graph Neural Network
- Transformer

Alemi, A. A et al, **DeepMath-Deep Sequence Models for Premise Selection**, *NIPS 2016*

Paliwal, A et al, **Graph Representations for Higher-Order Logic and Theorem Proving**, *AAAI 2020*

MAGNUSHAMMER: A TRANSFORMER-BASED APPROACH TO PREMISE SELECTION

Maciej Mikula*
Google DeepMind[†]

Szymon Tworkowski*
xAI[†]

Szymon Antoniak*
Mistral AI[†]

Bartosz Piotrowski
IDEAS NCBR

Albert Qiaochu Jiang
University of Cambridge

Jin Peng Zhou
Cornell University[‡]

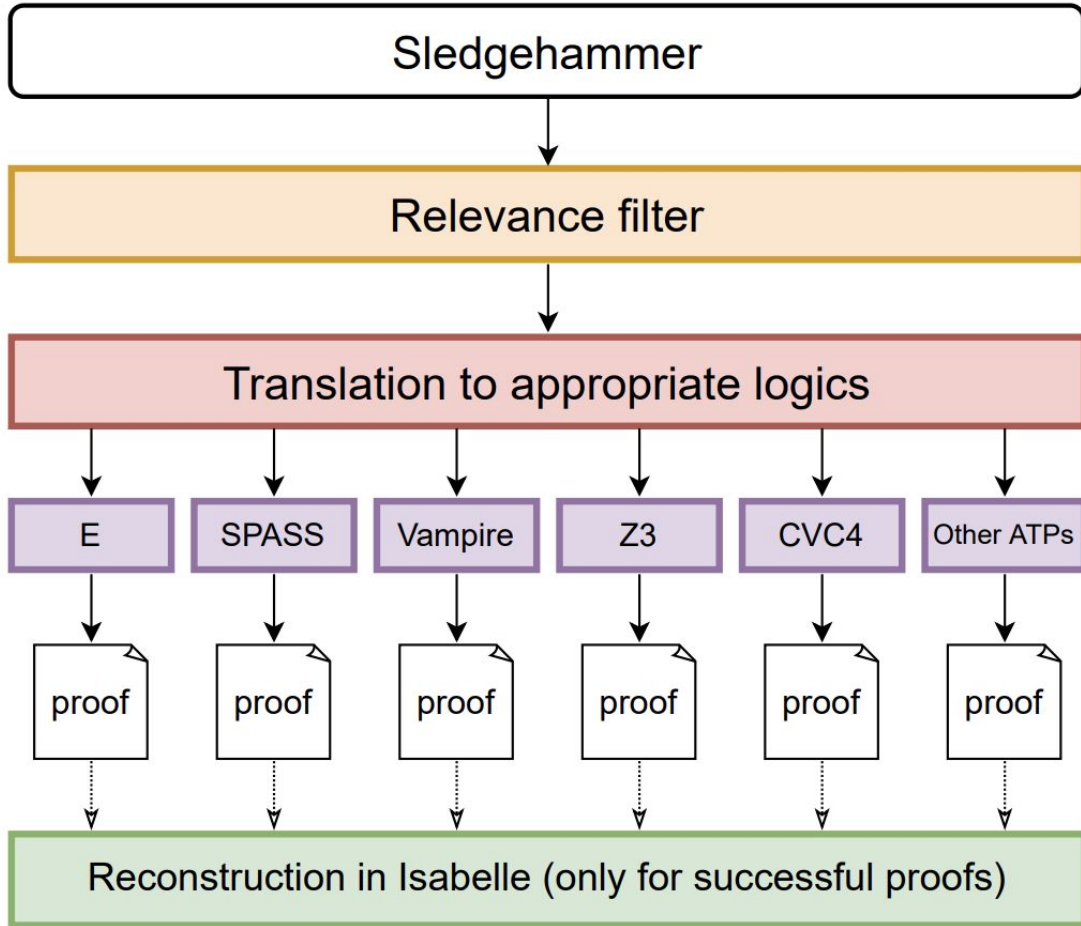
Christian Szegedy
xAI[‡]

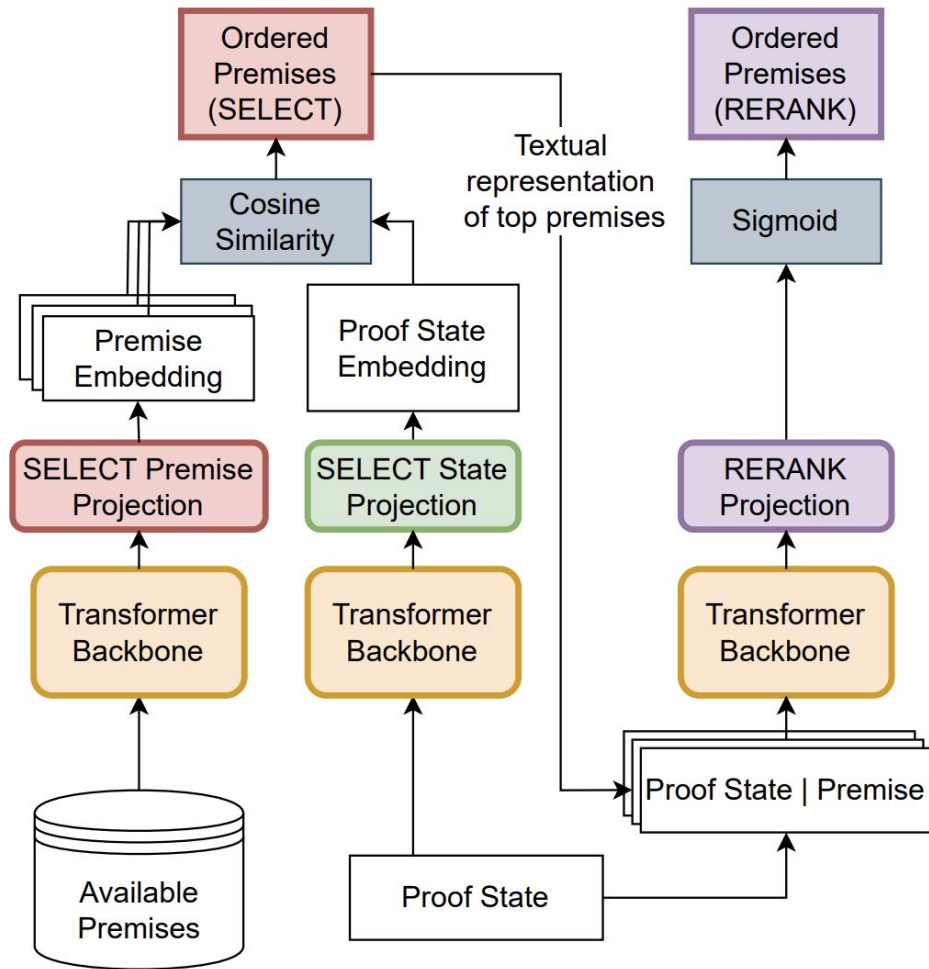
Łukasz Kuciński
IDEAS NCBR

Piotr Miłoś
IDEAS NCBR

Yuhuai Wu
xAI[‡]

ICLR 2024: Premise selection via Transformers for
Isabelle





Algorithm 1 Premise selection with Magnushammer.

Require:

`proof_state, premises` ▷ proof state to retrieve premises for and database of available premises
 `K_S, K_R` ▷ number of premises to retrieve with SELECT and RERANK, respectively

- 1: `state_embedding` \leftarrow `get_embeddings(proof_state)` ▷ SELECT stage starts
- 2: `premises_embeddings` \leftarrow `get_embeddings(premises)`
- 3: `Cache(premises_embeddings)`
- 4: `sim_scores` = `state_embedding · premises_embeddings`
- 5: `selected` = `premises[argsort(-sim_scores)[: K_S]]`
- 6: `batch` = [] ▷ RERANK stage starts
- 7: **for** `premise` in `selected` **do**
- 8: `batch.append((proof_state, premise))`
- 9: `rerank_scores` \leftarrow `get_rerank_scores(batch)`
- 10: `top_premises` = `selected[argsort(-rerank_scores)[: K_R]]`
- 11: **return** `top_premises`

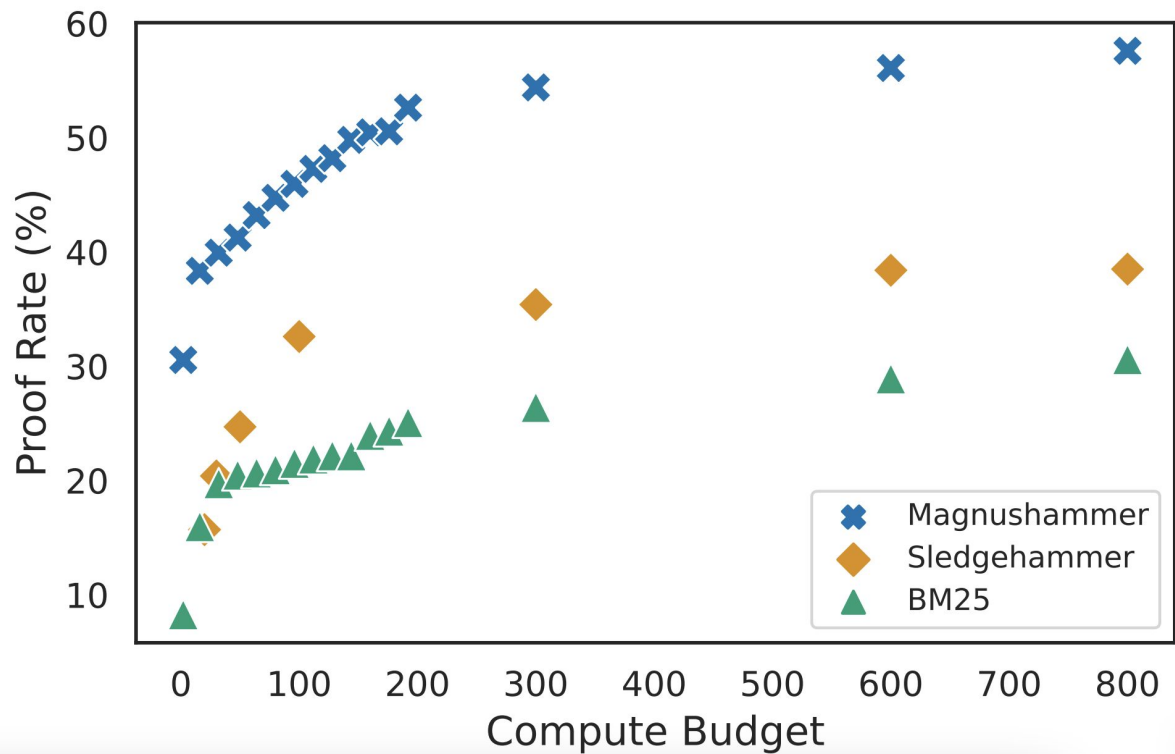
Results on PISA benchmark

Task	Method	Proof rate (%)
Single-step	BM25	30.6
	TF-IDF	31.8
	OpenAI embed. (Neelakantan et al., 2022)	36.1
	Sledgehammer	38.3
	Magnushammer (ours)	59.5
Multi-step	LISA (Jiang et al., 2021)	33.2
	Thor (Jiang et al., 2022a)	57.0
	Thor + Magnushammer (ours)	71.0

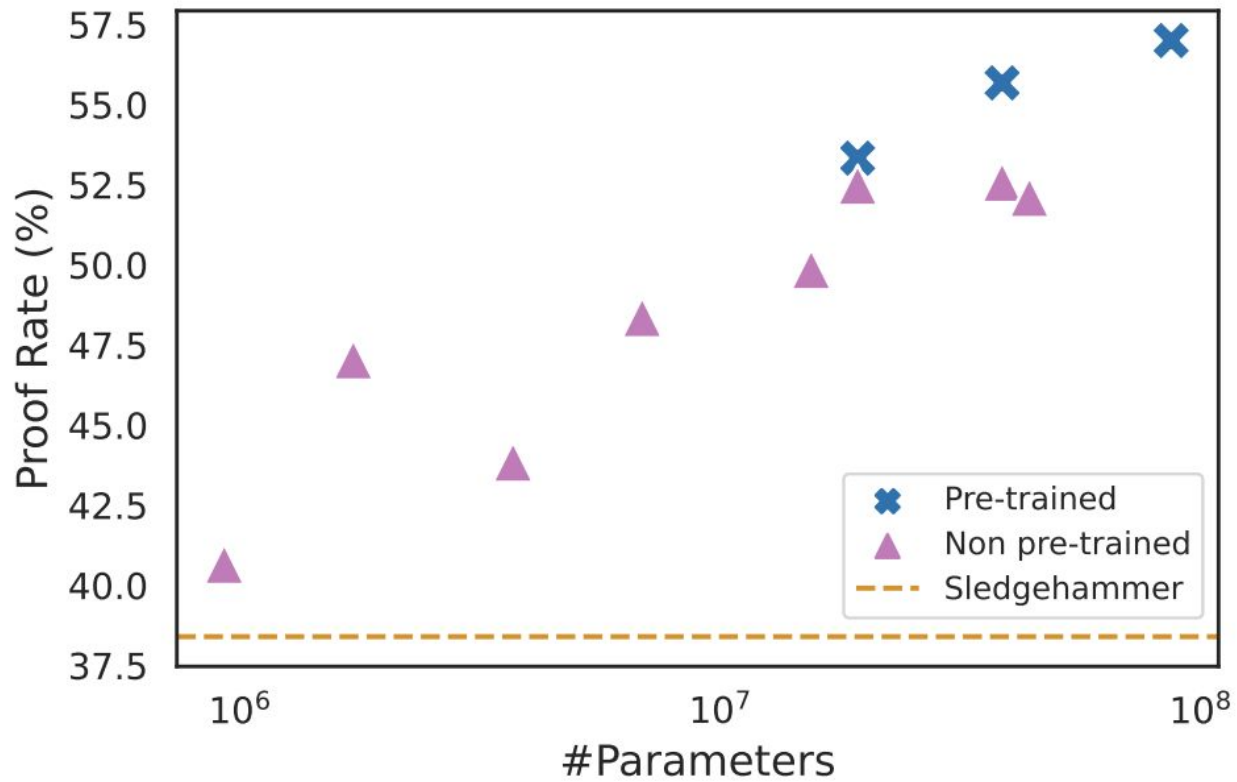
Results on MiniF2F

Task	Method	Valid (%)	Test (%)
Single-step	Sledgehammer	9.9	10.4
	Sledgehammer + heuristics	18.0	20.9
	Magnushammer (ours)	33.6	34.0
Multi-step	Thor + Sledgehammer (Jiang et al., 2022a)	28.3	29.9
	Thor + Sledgehammer + auto (Wu et al., 2022a)	37.3	35.2
	Thor + Magnushammer (ours)	36.9	37.3
	DSP (Jiang et al., 2022b)	43.9	39.3

Scalability



Scaling



The Vision of Autoformalization

