AI for Mathematics (AI4Math)

Paweł Balawender

Thursday, May 22, 2025

How can AI be used to reason about mathematics?

- You can prompt an LLM in natural language

How can AI be used to reason about mathematics?

- You can prompt an LLM in natural language
- But verifying the answer by a human is slow and error-prone!

How can AI be used to reason about mathematics?

- You can prompt an LLM in natural language
- But verifying the answer by a human is slow and error-prone!
- Mistakes in informal proofs are hard to spot

How can AI be used to reason about mathematics?

- You can prompt an LLM in natural language
- But verifying the answer by a human is slow and error-prone!
- Mistakes in informal proofs are hard to spot
- What if computers could verify reasoning?

How can AI be used to reason about mathematics?

- You can prompt an LLM in natural language
- But verifying the answer by a human is slow and error-prone!
- Mistakes in informal proofs are hard to spot
- What if computers could verify reasoning?
- **Takeaway 1**: we have ways to define abstract math concepts on a computer

How can AI be used to reason about mathematics?

- You can prompt an LLM in natural language
- But verifying the answer by a human is slow and error-prone!
- Mistakes in informal proofs are hard to spot
- What if computers could verify reasoning?
- **Takeaway 1**: we have ways to define abstract math concepts on a computer
- **Takeaway 2**: language models can generate detailed reasoning

How type systems come to help: C

- In the below example, the compiler can verify a simple reasoning

```c
int square(int x) { return x * x; }

int sq_of_4 = square(4);
```

How type systems come to help: C

- In the below example, the compiler can verify a simple reasoning
- Calling a function `int -> int` with an `int` argument results with an `int`

```
int square(int x) { return x * x; }

int sq_of_4 = square(4);
```

How type systems come to help: C++

- We like it when the type system is expressible, because we can offload checking a number of edge-cases to the compiler

```
std::option<PESEL> getPesel(Person& person);
auto pesel = getPesel(randomPerson);

// this fails - not everyone has PESEL assigned
print(pesel);
// this is ok
if (pesel) { print(pesel.value); }
```

How type systems come to help: Haskell

- Haskell allows us to define **inductive** types

```haskell
data Tree valT = Leaf valT | Node (Tree valT) (Tree valT)

size :: Tree valT -> Int
size (Leaf _) = 1
size (Node left right) = size left + size right
```

How type systems come to help: Rocq

- Rocq allows us to define **dependent** types

```
Inductive vec (valT : Type) : nat -> Type :=
| empty : vec valT 0
| append : forall n, valT -> vec valT n -> vec valT (n + 1)

(* equality type possible to define *)
Theorem lt_le_incl : forall n m, n < m -> n <= m.
```

How type systems come to help: Rocq

- Rocq allows us to define **dependent** types
- Here, we construct a family of types

```
Inductive vec (valT : Type) : nat -> Type :=
| empty : vec valT 0
| append : forall n, valT -> vec valT n -> vec valT (n + 1)

(* equality type possible to define *)
Theorem lt_le_incl : forall n m, n < m -> n <= m.
```

How type systems come to help: Rocq

- Rocq allows us to define **dependent** types
- Here, we construct a family of types
- For every natural number `n`, type of lists of length `n`

```
Inductive vec (valT : Type) : nat -> Type :=
| empty : vec valT 0
| append : forall n, valT -> vec valT n -> vec valT (n + 1)

(* equality type possible to define *)
Theorem lt_le_incl : forall n m, n < m -> n <= m.
```

How type systems come to help: Lean

- Lean: same theory as Rocq

How type systems come to help: Lean

- Lean: same theory as Rocq
- Lean community more liberal in accepting new axioms

How type systems come to help: Lean

- Lean: same theory as Rocq
- Lean community more liberal in accepting new axioms
- Take some type, e.g. set `{0, 1, 2, 3, ...}`

How type systems come to help: Lean

- Lean: same theory as Rocq
- Lean community more liberal in accepting new axioms
- Take some type, e.g. set `{0, 1, 2, 3, ...}`
- And some equivalence relation, e.g. `n -> n mod 2`

How type systems come to help: Lean

- Lean: same theory as Rocq
- Lean community more liberal in accepting new axioms
- Take some type, e.g. set `{0, 1, 2, 3, ...}`
- And some equivalence relation, e.g. `n -> n mod 2`
- The set `{0, 1}` of abstraction classes is the quotient

How type systems come to help: Lean

- Lean: same theory as Rocq
- Lean community more liberal in accepting new axioms
- Take some type, e.g. set `{0, 1, 2, 3, ...}`
- And some equivalence relation, e.g. `n -> n mod 2`
- The set `{0, 1}` of abstraction classes is the quotient
- Lean4 : allow taking a quotient of any type by any equivalence relation

How type systems come to help: Lean

- Lean: same theory as Rocq
- Lean community more liberal in accepting new axioms
- Take some type, e.g. set `{0, 1, 2, 3, ...}`
- And some equivalence relation, e.g. `n -> n mod 2`
- The set `{0, 1}` of abstraction classes is the quotient
- Lean4 : allow taking a quotient of any type by any equivalence relation
- This is called **quotient types**

How type systems come to help: Lean

- Lean: same theory as Rocq
- Lean community more liberal in accepting new axioms
- Take some type, e.g. set `{0, 1, 2, 3, ...}`
- And some equivalence relation, e.g. `n -> n mod 2`
- The set `{0, 1}` of abstraction classes is the quotient
- Lean4 : allow taking a quotient of any type by any equivalence relation
- This is called **quotient types**
- Enabled Lean to have one, standard library for real analysis

Lean readily models ZFC set theory

```
structure Setoid (type : Type) :=
    (relation : type -> type -> Prop)
    (proofEquivalenceRelation : isEquivalence relation)

-- elements of type `Set type` are equivalence classes
-- this leads to implementation of ZFC set theory
def Set (type : Type) [s : Setoid type] : Type :=
    Quotient s relation
```

Putnam Bench

- Benchmark for theorem-proving algorithms

Putnam Bench

- Benchmark for theorem-proving algorithms
- Problems from Putnam Mathematical Competition years 1962-2023

Putnam Bench

- Benchmark for theorem-proving algorithms
- Problems from Putnam Mathematical Competition years 1962-2023
- 657 problems formalized in Lean 4

Putnam Bench

- Benchmark for theorem-proving algorithms
- Problems from Putnam Mathematical Competition years 1962-2023
- 657 problems formalized in Lean 4
- 640 in Isabelle

Putnam Bench

- Benchmark for theorem-proving algorithms
- Problems from Putnam Mathematical Competition years 1962-2023
- 657 problems formalized in Lean 4
- 640 in Isabelle
- 412 in Rocq

Putnam Bench

- Benchmark for theorem-proving algorithms
- Problems from Putnam Mathematical Competition years 1962-2023
- 657 problems formalized in Lean 4
- 640 in Isabelle
- 412 in Rocq
- 253 problems in algebra

Putnam Bench

- Benchmark for theorem-proving algorithms
- Problems from Putnam Mathematical Competition years 1962-2023
- 657 problems formalized in Lean 4
- 640 in Isabelle
- 412 in Rocq
- 253 problems in algebra
- 226 in analysis

Putnam Bench

- Benchmark for theorem-proving algorithms
- Problems from Putnam Mathematical Competition years 1962-2023
- 657 problems formalized in Lean 4
- 640 in Isabelle
- 412 in Rocq
- 253 problems in algebra
- 226 in analysis
- 108 in number theory

Putnam Bench

- Benchmark for theorem-proving algorithms
- Problems from Putnam Mathematical Competition years 1962-2023
- 657 problems formalized in Lean 4
- 640 in Isabelle
- 412 in Rocq
- 253 problems in algebra
- 226 in analysis
- 108 in number theory
- 10 in probability etc.

Putnam 1962 A1, informal

- Given five points in a plane, no three of which lie on a straight line, show that some four of these points form the vertices of a convex quadrilateral.

Putnam 1962 A2, formal

```
From mathcomp Require Import all_algebra all_ssreflect.
From mathcomp Require Import reals normedtype sequences topology derive measure lebesgue_measure lebesgue_in
From mathcomp Require Import classical_sets.

Set Implicit Arguments.
Unset Strict Implicit.
Unset Printing Implicit Defensive.

Local Open Scope ring_scope.
Local Open Scope classical_set_scope.

Variable R : realType.
Definition mu := [the measure _ _ of @lebesgue_measure R].
Definition putnam_1962_a2_solution : set (R -> R) := [set f | exists a c : R, a >= 0 /\ f = (fun x : R => a
Theorem putnam_1962_a2
    (P : (set R) -> (R -> R) -> Prop)
    (P_def : forall s f, P s f <-> ((forall x, f x >= 0) /\ forall x, x \in s ->
                1/x * \int[mu]_(t in [set t | 0 <= t <= x]) f t = Num.sqrt (f 0 * f x)))
    : (forall f,
        (P [set t | 0 < t] f -> exists g, g \in putnam_1962_a2_solution /\ (forall x : R, x > 0 -> f x = g x
        (forall e, 0 < e -> P [set t | 0 < t < e] f -> exists g, g \in putnam_1962_a2_solution /\ (forall x
        forall f, f \in putnam_1962_a2_solution -> P [set t | 0 < t] f \/ exists e, 0 < e /\ P [set t | 0 <
Proof. Admitted.
```

DeepSeek-Prover-V2

- submitted to arXiv on 30 Apr 2025

DeepSeek-Prover-V2

- submitted to arXiv on 30 Apr 2025
- previous SOTA on Putnam Bench: Kimina-Prover-7B-Disill

DeepSeek-Prover-V2

- submitted to arXiv on 30 Apr 2025
- previous SOTA on Putnam Bench: Kimina-Prover-7B-Disill
- 10 problems out of 657 (Lean)

DeepSeek-Prover-V2

- submitted to arXiv on 30 Apr 2025
- previous SOTA on Putnam Bench: Kimina-Prover-7B-Disill
- 10 problems out of 657 (Lean)
- DeepSeek-Prover-V2-671B claims: 49

DeepSeek-Prover-V2

- submitted to arXiv on 30 Apr 2025
- previous SOTA on Putnam Bench: Kimina-Prover-7B-Disill
- 10 problems out of 657 (Lean)
- DeepSeek-Prover-V2-671B claims: 49
- problems involved; some proofs “cheat”

How DeepSeek prover works

- Prompt general-purpose LLM for natural language proof sketch

How DeepSeek prover works

- Prompt general-purpose LLM for natural language proof sketch
- Here: DeepSeek-V3, Chain of Thought mode

How DeepSeek prover works

- Prompt general-purpose LLM for natural language proof sketch
- Here: DeepSeek-V3, Chain of Thought mode
- Also with DeepSeek-V3, translate subgoals to Lean 4 unproved theorems

How DeepSeek prover works

- Prompt general-purpose LLM for natural language proof sketch
- Here: DeepSeek-V3, Chain of Thought mode
- Also with DeepSeek-V3, translate subgoals to Lean 4 unproved theorems
- Smaller 7B prover model to complete Lean proofs of subgoals

How DeepSeek prover works

- Prompt general-purpose LLM for natural language proof sketch
- Here: DeepSeek-V3, Chain of Thought mode
- Also with DeepSeek-V3, translate subgoals to Lean 4 unproved theorems
- Smaller 7B prover model to complete Lean proofs of subgoals
- Use reinforcement learning to train 7B prover

How DeepSeek prover works

- Prompt general-purpose LLM for natural language proof sketch
- Here: DeepSeek-V3, Chain of Thought mode
- Also with DeepSeek-V3, translate subgoals to Lean 4 unproved theorems
- Smaller 7B prover model to complete Lean proofs of subgoals
- Use reinforcement learning to train 7B prover
- Binary reward: proof finished / unfinished.

How DeepSeek prover works

- Prompt general-purpose LLM for natural language proof sketch
- Here: DeepSeek-V3, Chain of Thought mode
- Also with DeepSeek-V3, translate subgoals to Lean 4 unproved theorems
- Smaller 7B prover model to complete Lean proofs of subgoals
- Use reinforcement learning to train 7B prover
- Binary reward: proof finished / unfinished.
- This requires Rocq / Lean to check the proof

How DeepSeek prover works

- Prompt general-purpose LLM for natural language proof sketch
- Here: DeepSeek-V3, Chain of Thought mode
- Also with DeepSeek-V3, translate subgoals to Lean 4 unproved theorems
- Smaller 7B prover model to complete Lean proofs of subgoals
- Use reinforcement learning to train 7B prover
- Binary reward: proof finished / unfinished.
- This requires Rocq / Lean to check the proof
- RL algorithm: Group Relative Policy Optimization (GRPO): not in scope

AI4Math research in Warsaw: Magnushammer

MAGNUSHAMMER: A TRANSFORMER-BASED APPROACH TO PREMISE SELECTION

Maciej Mikuła* 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‡

ABSTRACT

This paper presents a novel approach to premise selection, a crucial reasoning task in automated theorem proving. Traditionally, symbolic methods that rely on extensive domain knowledge and engineering effort are applied to this task. In contrast, this work demonstrates that contrastive training with the transformer architecture can achieve higher-quality retrieval of relevant premises, without the engineering overhead. Our method, Magnushammer, outperforms the most advanced and widely used automation tool in interactive theorem proving called Sledgehammer. On the PISA and miniF2F benchmarks Magnushammer achieves 59.5% (against 38.3%) and 34.0% (against 20.9%) success rates, respectively. By combining Magnushammer with a language-model-based automated theorem prover, we further improve the state-of-the-art proof success rate from 57.0% to 71.0% on the PISA benchmark using 4x fewer parameters. Moreover, we develop and open source a