The Last Mile

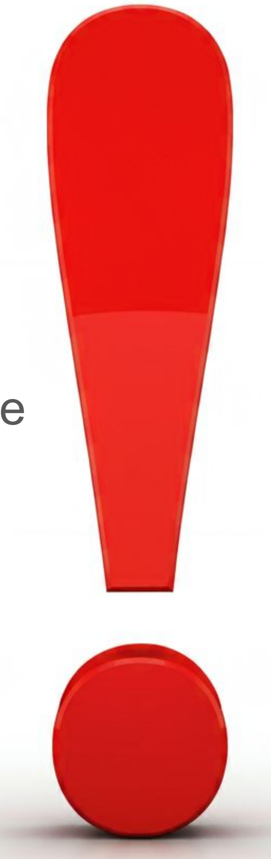
How do we make AI theorem provers
which work in the real world for real users
and not just on benchmarks?

Jason Rute
IBM Research / MIT-IBM Watson AI Lab

AI Generated Image (ImageFX)

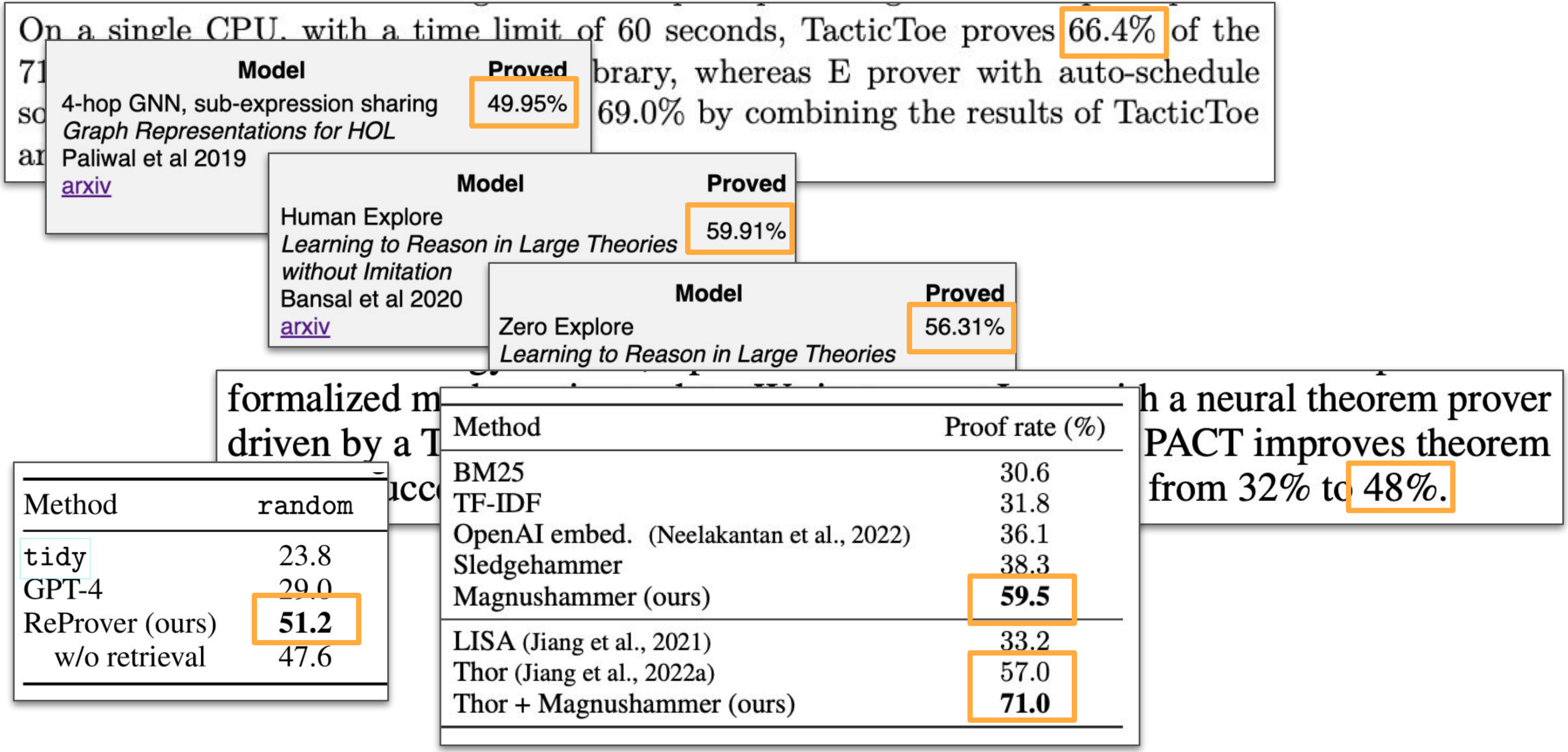
Disclaimers

- This talk is intentionally provocative
- The opinions are my own, not those of my employer or co-authors
- The ideas are not necessarily novel, and others have said them before
- I could be wrong about the solutions … or even the problems
- I'm one of the worst offenders
- I started working on these slides yesterday
- …but I've been thinking about this stuff for a long while!



AI Generated Image (ImageFX)

AI solves 48%+ of HOL/Lean/Isabelle library theorems



On a single CPU, with a time limit of 60 seconds, TacticToe proves 66.4% of the

71 brary, whereas E prover with auto-schedule

so 69.0% by combining the results of TacticToe

ar

Model	Proved
4-hop GNN, sub-expression sharing *Graph Representations for HOL* Paliwal et al 2019 arxiv	49.95%

Model	Proved
Human Explore *Learning to Reason in Large Theories without Imitation* Bansal et al 2020 arxiv	59.91%

Model	Proved
Zero Explore *Learning to Reason in Large Theories*	56.31%

formalized m h a neural theorem prover

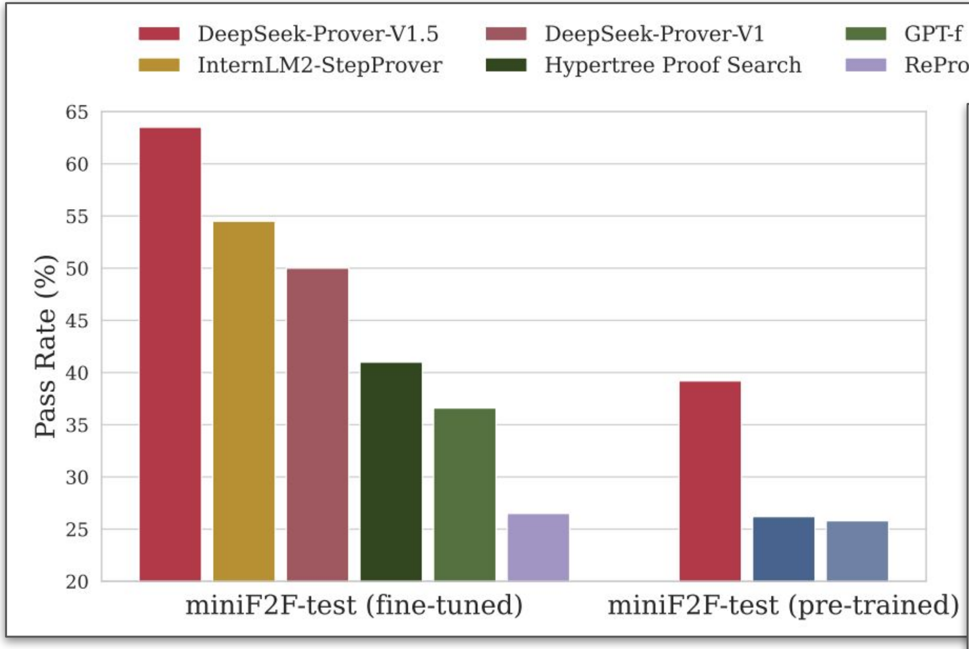
driven by a T PACT improves theorem

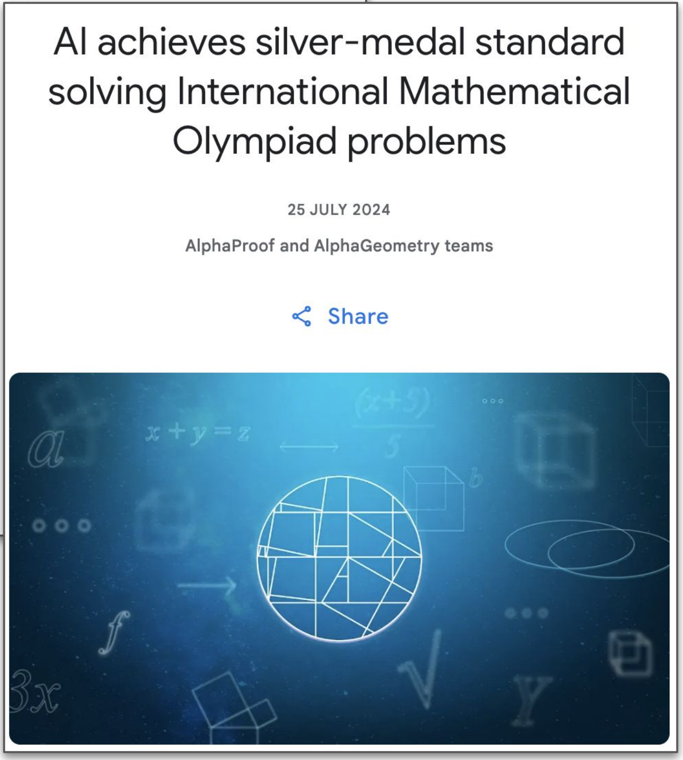
ucc from 32% to 48%.

Method	random
tidy	23.8
GPT-4	29.0
ReProver (ours)	**51.2**
w/o retrieval	47.6

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

AI can solve competition math problems





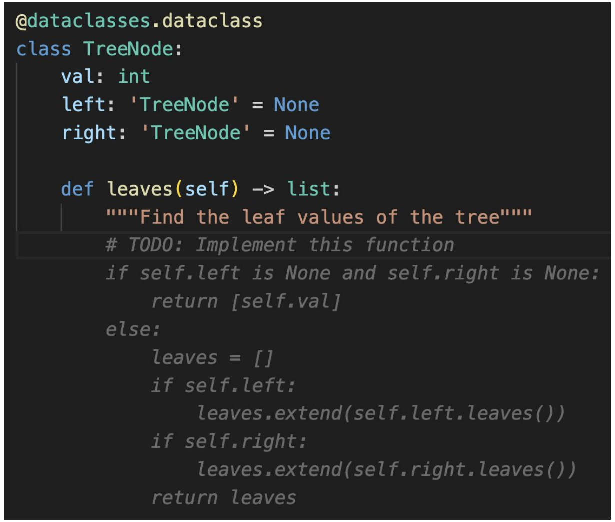
AI achieves silver-medal standard solving International Mathematical Olympiad problems

25 JULY 2024

AlphaProof and AlphaGeometry teams

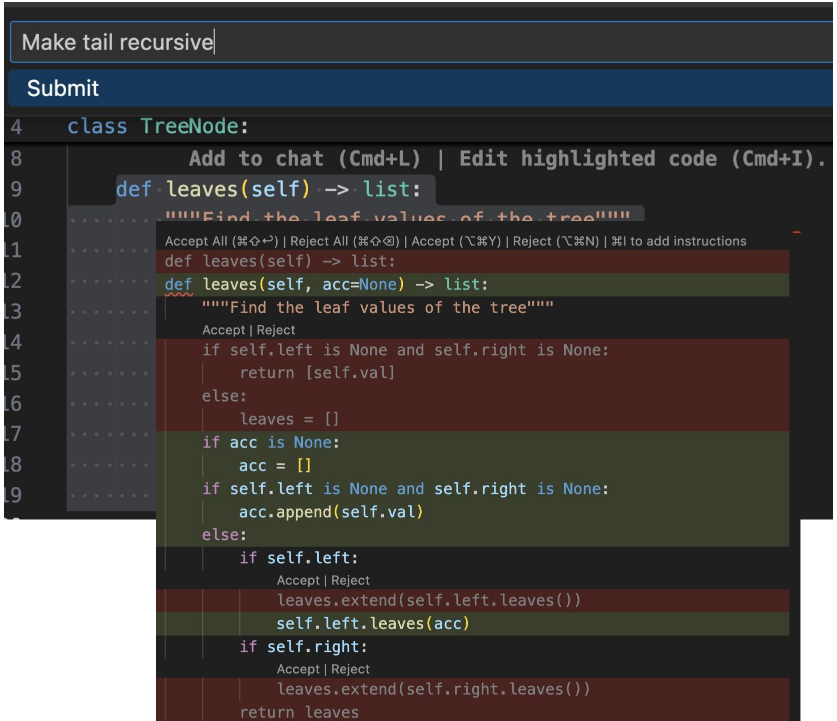
Code assistants are becoming quite popular and good

Suggest Code



```python
@dataclasses.dataclass
class TreeNode:
    val: int
    left: 'TreeNode' = None
    right: 'TreeNode' = None

    def leaves(self) -> list:
        """Find the leaf values of the tree"""
        # TODO: Implement this function
        if self.left is None and self.right is None:
            return [self.val]
        else:
            leaves = []
            if self.left:
                leaves.extend(self.left.leaves())
            if self.right:
                leaves.extend(self.right.leaves())
            return leaves
```

Edit Code



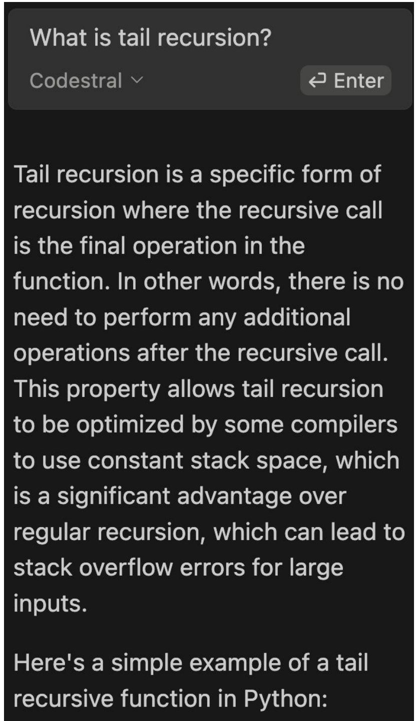
Make tail recursive
Submit

```
class TreeNode:
        Add to chat (Cmd+L) | Edit highlighted code (Cmd+I).
    def leaves(self) -> list:
        """Find the leaf values of the tree"""
```

Accept All (⌘⇧↩) | Reject All (⌘⇧⌫) | Accept (⌥⌘Y) | Reject (⌥⌘N) | ⌘I to add instructions

```python
def leaves(self) -> list:
def leaves(self, acc=None) -> list:
    """Find the leaf values of the tree"""
    Accept | Reject
    if self.left is None and self.right is None:
        return [self.val]
    else:
        leaves = []
    if acc is None:
        acc = []
    if self.left is None and self.right is None:
        acc.append(self.val)
    else:
        if self.left:
            Accept | Reject
            leaves.extend(self.left.leaves())
            self.left.leaves(acc)
        if self.right:
            Accept | Reject
            leaves.extend(self.right.leaves())
        return leaves
```

Chat



What is tail recursion?

Tail recursion is a specific form of recursion where the recursive call is the final operation in the function. In other words, there is no need to perform any additional operations after the recursive call. This property allows tail recursion to be optimized by some compilers to use constant stack space, which is a significant advantage over regular recursion, which can lead to stack overflow errors for large inputs.

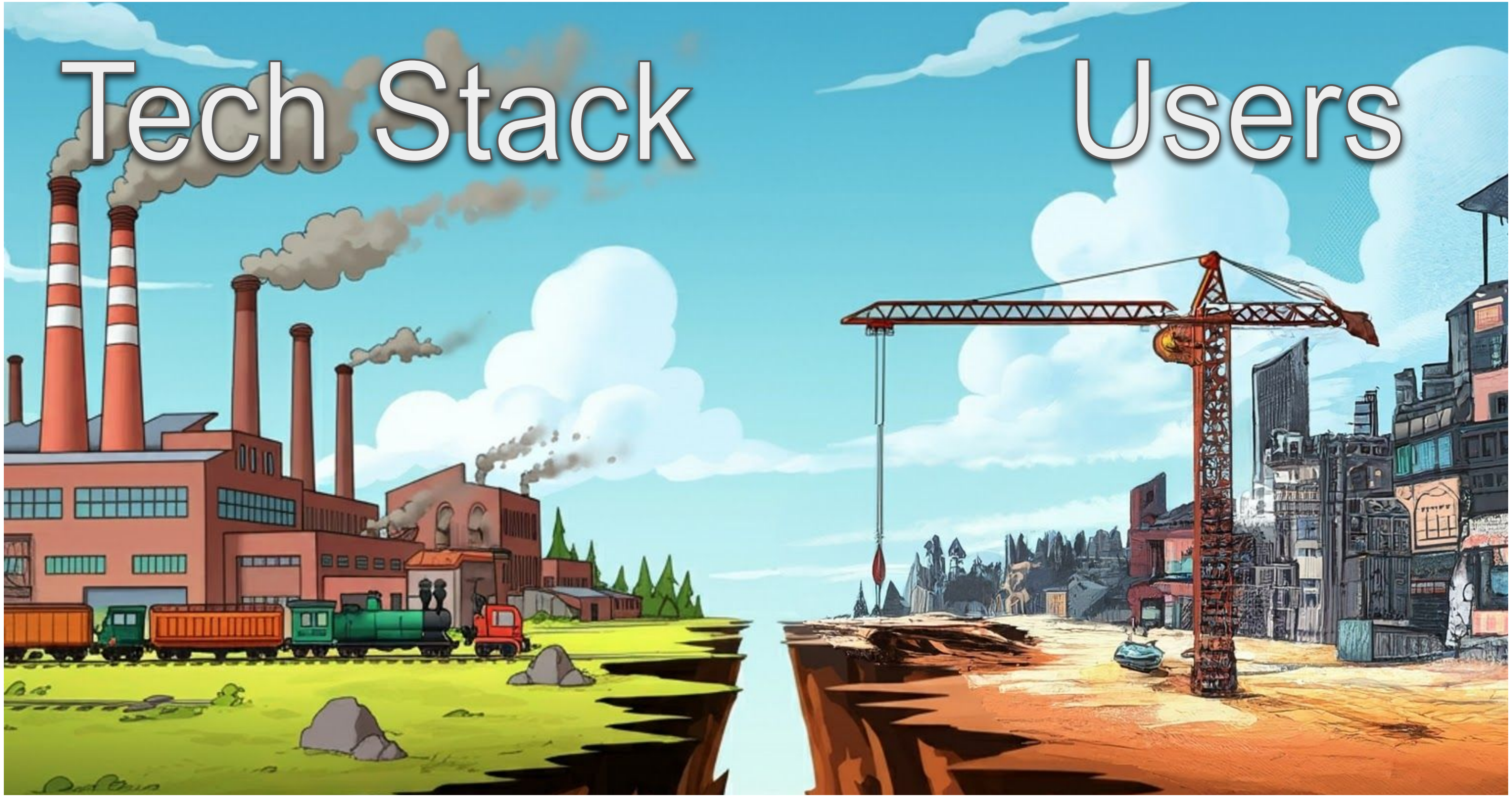
Here's a simple example of a tail recursive function in Python:

Why don't we have AI assistants for real ITP users?

- Yes we have hammers.
- But why don't we have more?
- Why hasn't the feedback to Lean Copilot, LLMLean, Tactician, CoqPilot, etc been positive?
- Is it because the technology is not good enough yet?

- Claim: The technology is basically good enough, but we aren't building the right sorts of assistants.

The Last Mile



AI Generated Image (ImageFX)

Problems

Parties involved

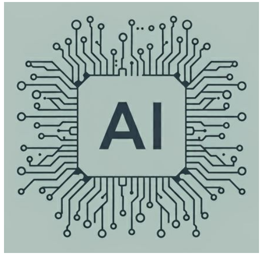
Users

Novice

Expert

ITP Governing Bodies

AI Labs



Industry

Academia

Startups

User Stories: Figuring out what users want

Alice is a new ITP user. She is confused where to start. She is very frustrated that she can't even prove that (xy + yz) / y = x + z. She wants her code editor to suggest next steps, and maybe also a built-in chat bot.

Bob is an ITP power user. He can prove 10 lemmas before breakfast, but there is so much work to do! He wants tools to make him 10x more efficient. This includes proof automation, but also general purpose coding support.

Catherine is working on a long term formalization project. She wants tools which will run overnight on a server, closing simple goals, learning from the current code base, and even formalizing parts of her LaTeX blueprint.

Doug is a mathematician exploring new ideas in his field. He is formatting conjectures that he would love to be able to prove or disprove automatically with AI. He is willing to be a guinea pig, partnering with AI experts.

User experience shouldn't take backdoor to benchmarks

Bad UX (Inconvenience > Benefit)

Researchers open sourced their SoTA model and interface code. The installation is a huge pain, so much that most never try it. They provide a model to run locally. It heats up the laptop and barely solves anything in a minute. They also provide a way to use an LLM API, but one can quickly rack up $100 in cost in the first day. No one adopts it.

Good UX (Benefit > Inconvenience)

An ITP ships an AI assistant as part of the standard setup of its system. It is easy to setup with good docs and tech support. It's been designed with user experience in mind. It is far from SoTA, but it provides good, unobtrusive advice when it works. To entice users, there is a free web version to try it out, and the ITP power users and maintainers advertise it.

ITP Benchmarks are **All** Broken

- Library theorems
  - **Practical Real-world proofs**
  - Difficult to compare different AIs
  - Can easily be memorized by LLMs
  - Unrealistic random split of the library
- Mini-F2F
  - **Cross platform**
  - Problems avoid practical concerns
  - LLM data leakage is a concern
  - Has become a "target" a la Goodhart's law
- IMO Grand Challenge
  - **Great for measuring SoTA**
  - Little to do with typical ITP experience

- End result: We don't know what works

- What is missing from benchmarks?
  - Real ITP problems that users want AI to do
  - Program verification
  - Participation of symbolic AI (e.g. Hammers)
  - Participation of practical ITP AI systems
  - LLM-memorization resistant problems
  - Regular new problems (annually?)
  - Tracking of computational resources used
  - Good plan for cross platform benchmarks
  - Benchmarks other than solving theorems
  - Better versioning and governance

Lack of Communication

- AI researchers and ITP maintainers don't coordinate enough
  - This leads to reinventing the wheel
  - AI systems can't be incorporated into real world systems
  - AI experiments are done on forked version of the ITP
- Startups and ITP maintainers don't coordinate enough
  - Startups just drop stuff
- Symbolic AI and Neural AI researchers don't coordinate enough
  - AI papers consistently beat Hammers on AI paper benchmarks, but no response from Hammer folks
  - No common benchmarks by neural and symbolic AI folks
  - Lean is still investing a lot into building a Hammer
- Almost no one is communicating with users
  - Users should be more involved in making of AI benchmarks
  - User experience should be front and center in design of AI tools

How Research Papers "Cheat"

- Use ridiculous amounts of compute for training and inference

- Precompute and hard-code lemmas for lemma selection

- Training data is unrealistically close to test data (random split)

Technical Challenges and Shortcuts

- How do we keep lemma selection database up-to-date
  - Continual retraining (e.g. in CI)
  - Online updates
- Do I host model locally on machine or in the cloud via and API?
  - API is easier and more powerful, but expensive
  - Local isn't at mercy of developer
- How to make an easy-to-install system?
- How to keep an AI system up-to-date as the library changes?
  - Online learning and/or lemma selection? (Hammer, Tactician, Graph2Tac, Reprover, miniCTX)
  - Continual retraining?
- Incorporate proof search (usually done with tactics) with surrounding code (usually done in the editor)

Proposed Solutions

Principles

- If you have a vision, build it!
- Start simple, **get user feedback**, and improve
- Advertise your system a lot
- **Focus on the user experience first and foremost**
- Don't let perfect be the enemy of good enough
- Measure progress
- Push all common components back into common tools
  - Language agnostic stuff shouldn't be tied to a specific ITP
  - Model agnostic stuff (like search) shouldn't be tied to a particular model
- Don't pigeonhole yourself
- Rapid improvement

Hammers

- Hammers work in practice!
- Isabelle, Coq, and (soon) Lean have them
- Need engagement between Neural AI and Hammer folks
    - Are the current hammer benchmarks good?
    - What do hammer folks think of neural AI tools?
- What has made hammers so successful that other AI tools can learn from?

Lemma Selection

- Universal feature
    - Used in hammers
    - Used in retrieval augmented generation
- ITPs need to solve how to do vector lookup of premises from the library
    - How to make efficient (speed and memory)
    - How small of embeddings can you have
    - How to precompute embeddings during CI
    - How to bulk compute embeddings for a local project
    - How to compute embeddings on the fly for new theorems
    - Should it be done in pure OCaml/C++/Lean or in a neural network?

Magnus Hammer

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‡

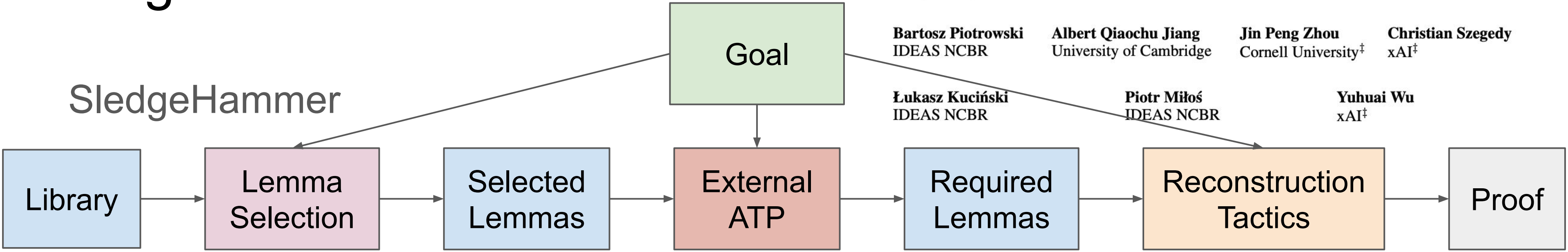
Christian Szegedy
xAI‡

Łukasz Kuciński
IDEAS NCBR

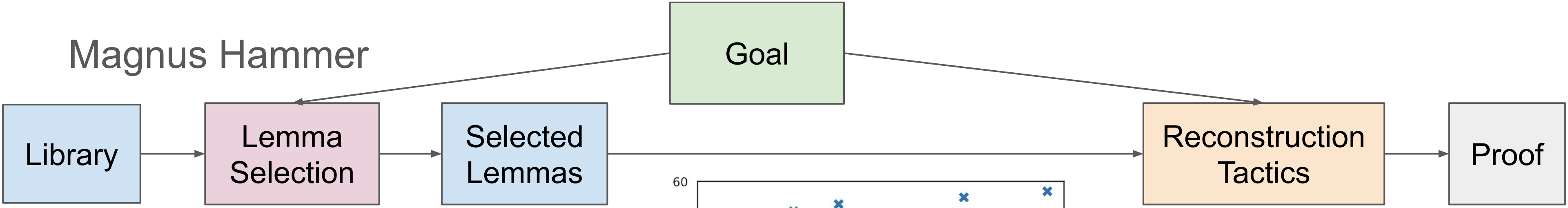
Piotr Miłoś
IDEAS NCBR

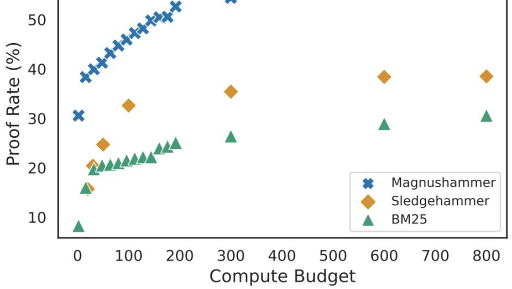
Yuhuai Wu
xAI‡

SledgeHammer



Magnus Hammer





k-NN (TacticToe, HOList, Tactician)

- Record hand-crafted features for each proofstate-goal paper
- Use k-NN to look up closest proofstates
- Copy those tactics (verbatim)
- Tree search with selected tactics
- (Optional) Use online k-NN to take into account recent information
- (Optional) Use locality sensitive hashing forest to search over all tactics

- Works much better than you think!
- Should be a standard baseline in all AI for TP papers!
- Can be written (mostly) as a pure tactic.

Code assistants

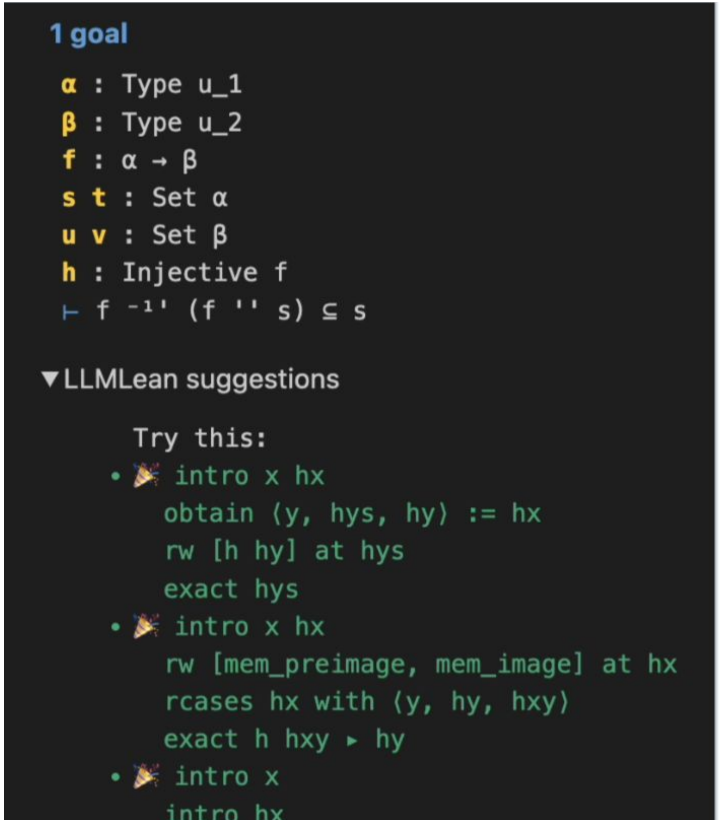
- We should test and benchmark general purpose coding assistants like we do ITP specific tools.
- Do they help with day-to-day tasks?
- What are they good and not good at?
- Where do they fail?
- What community advice do we have for better prompting?

Need practical benchmarks

- We need benchmarks for real world theorem provers
- Need to benchmark real systems on real hardware on real problems
- Should benchmark:
  - Built-in general purpose tactics
  - Hammers
  - Code assistant models
  - ITP specific tools
- Every practical tool should be measured on two criteria:
  - User feedback
  - Benchmarks
- If these aren't aligned, we need to modify the benchmarks
- ITP users should be actively involved in making real benchmark problems

Some things I like: "Try this:" in Lean

- Click on tactic to replace tactic call with code
- User-centric feature
- Common tool available to other tactics
- Used in:
  - Standard Mathlib tactics
  - Aesop
  - Lean Copilot
  - LLMLean

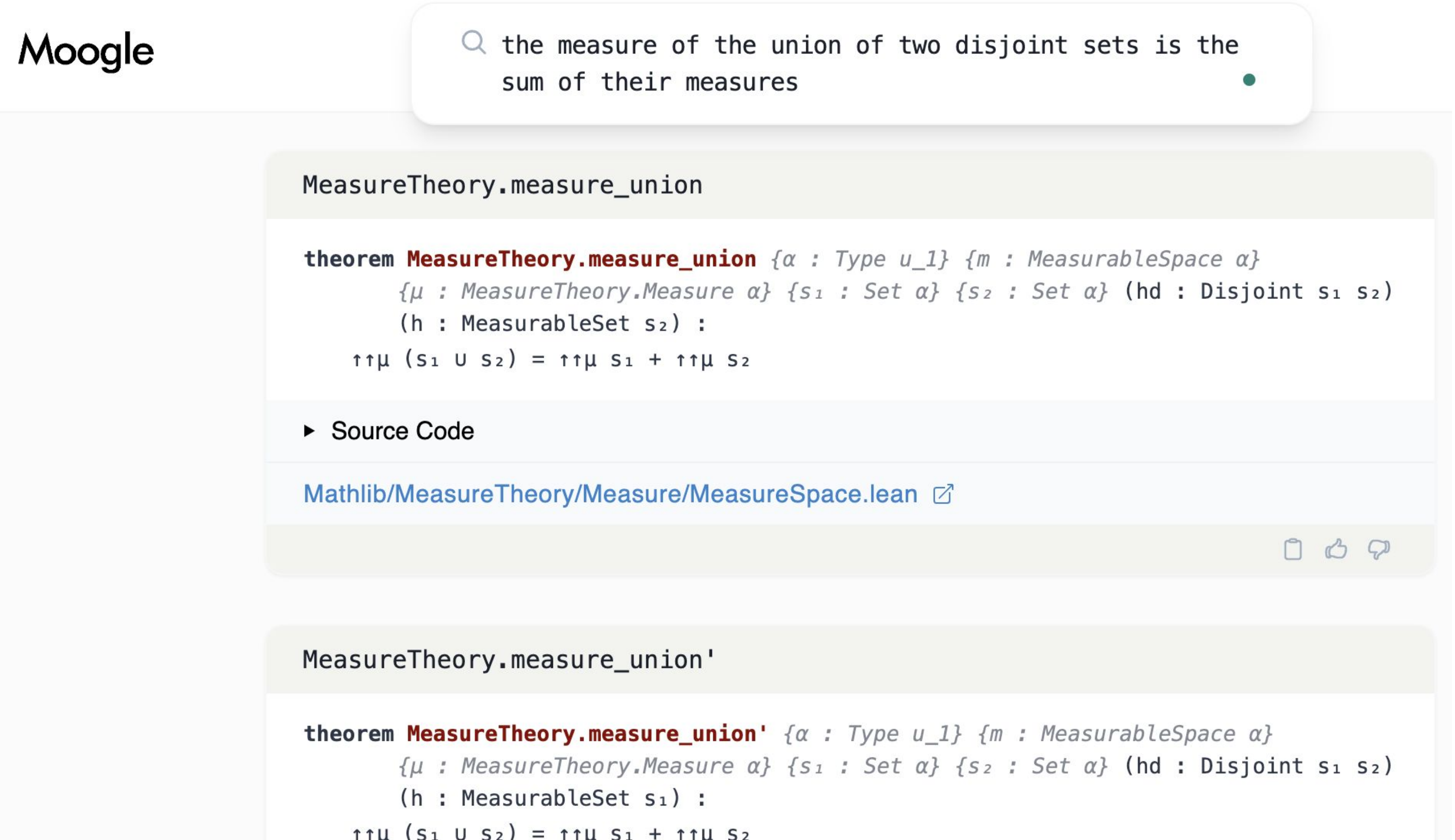


```
1 goal
 α : Type u_1
 β : Type u_2
 f : α → β
 s t : Set α
 u v : Set β
 h : Injective f
 ⊢ f ⁻¹' (f '' s) ⊆ s

▼LLMLean suggestions

      Try this:
    • 🎉 intro x hx
        obtain ⟨y, hys, hy⟩ := hx
        rw [h hy] at hys
        exact hys
    • 🎉 intro x hx
        rw [mem_preimage, mem_image] at hx
        rcases hx with ⟨y, hy, hxy⟩
        exact h hxy ▸ hy
    • 🎉 intro x
        intro hx
```

Some things I like: Aesop

- Highly customizable search in Lean
- Available for other AI tools
- Used by Lean Copilot
- Easy to hook up your AI model to a Lean proof search

Some things I like: Moogle.ai



the measure of the union of two disjoint sets is the sum of their measures

MeasureTheory.measure_union

```
theorem MeasureTheory.measure_union {α : Type u_1} {m : MeasurableSpace α}
      {μ : MeasureTheory.Measure α} {s₁ : Set α} {s₂ : Set α} (hd : Disjoint s₁ s₂)
      (h : MeasurableSet s₂) :
   ↑↑μ (s₁ ∪ s₂) = ↑↑μ s₁ + ↑↑μ s₂
```

▸ Source Code

Mathlib/MeasureTheory/Measure/MeasureSpace.lean

MeasureTheory.measure_union'

```
theorem MeasureTheory.measure_union' {α : Type u_1} {m : MeasurableSpace α}
      {μ : MeasureTheory.Measure α} {s₁ : Set α} {s₂ : Set α} (hd : Disjoint s₁ s₂)
      (h : MeasurableSet s₁) :
   ↑↑μ (s₁ ∪ s₂) = ↑↑μ s₁ + ↑↑μ s₂
```

Proposal: ITP AI Influencer

- Figures out what ITP users really want out of AI systems
- Collects benchmark problems
- Benchmarks current systems in real world settings
- Advocates for development of common technologies and interfaces useful to many AI systems so don't need to reinvent the wheel
- When new AI systems comes out:
  - Advertises them
  - Gives honest assessments of their strengths and weaknesses and use cases
  - Collects user feedback
  - Works with system maintainers and ITP maintainers to upstream common useful components
  - Works with system maintainers to make the next interaction better