LLM Reasoning for Math & Code

Peiyang Song
California Institute of Technology
psong@caltech.edu

Peiyang Song (*website*, *psong@caltech.edu*)

- 4th-year undergrad @ **Caltech**, Major in **CS** + Minor in **Robotics**
  - Advisors: Prof. Steven Low & Prof. Günter Niemeyer
- A “sequential” overview of my research journey: my 4 research internships
  - UCSB Computer Architecture Lab (ArchLab) with Prof. **Tim Sherwood** & Dr. **Jeremy Lau**
  - Caltech Anima AI + Science Lab with Prof. **Anima Anandkumar** & Dr. **Kaiyu Yang**
  - Stanford AI Lab (SAIL) with Prof. **Noah Goodman** & Dr. **Gabriel Poesia**
  - Berkeley AI Research (BAIR) Lab with Prof. **Dawn Song** & Dr. **Jingxuan He**

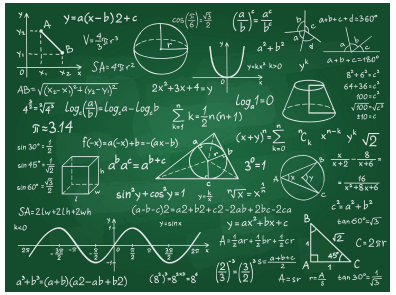
Applying for CS-related PhD positions this current cycle (26 Fall)

Research Interest

Keywords: **LLM reasoning**, **agentic AI**, **neuro-symbolic AI**.

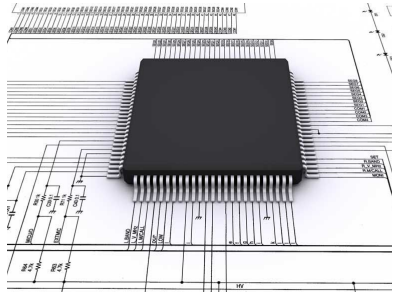
- One **central axis** (**AI for Math & Code**): Integrating LLM reasoning, agentic AI, and neuro-symbolic methods by combining *neural* models (LLMs) with formal *symbolic* systems (such as *Lean*) to advance LLM-based agents for formal reasoning in *math* and *code*.
- Broader **LLM Reasoning**: Extending LLM reasoning beyond formal domains, exploring how LLMs can tackle *informal* reasoning in natural language, inspired by *cognitive science* principles and studies of *human-like* reasoning.
- Broader **Neuro-Symbolic AI**: Building general *neuro-symbolic* approaches towards *fundamental* AI systems beyond formal reasoning, such as developing *energy-efficient inference* and *machine translation* across idioms and languages.

Automated Reasoning and Formal Proofs



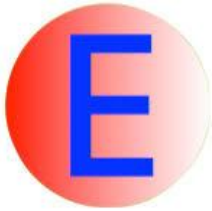
Formal mathematics

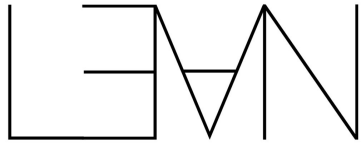
Software verification



Hardware verification

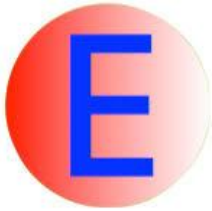
Cyber-physical systems





Automated Reasoning and Formal Proofs

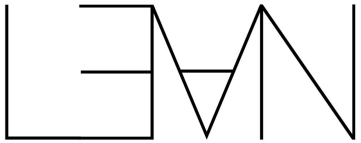
- **Automated theorem proving**
  - SMT solvers, model checkers, ATP systems in first-order logic, etc.
  - Minimal efforts from humans
  - Limited expressiveness
  - Difficult to scale



Automated Reasoning and Formal Proofs

- **Interactive theorem proving**
  - Proof assistants such as Coq, Isabelle, and Lean
  - Expressive logic, e.g., dependent type theory
  - Successfully used in large formalization projects
  - **Lots of efforts from humans to write proofs**
  - **Proof automation is critical for wider adoption**

[Hales et al., "A Formal Proof of the Kepler Conjecture", 2017]
[Klein et al., "seL4: Formal Verification of an OS Kernel", 2009]
[Leroy, "Formal Verification of a Realistic Compiler", 2008]

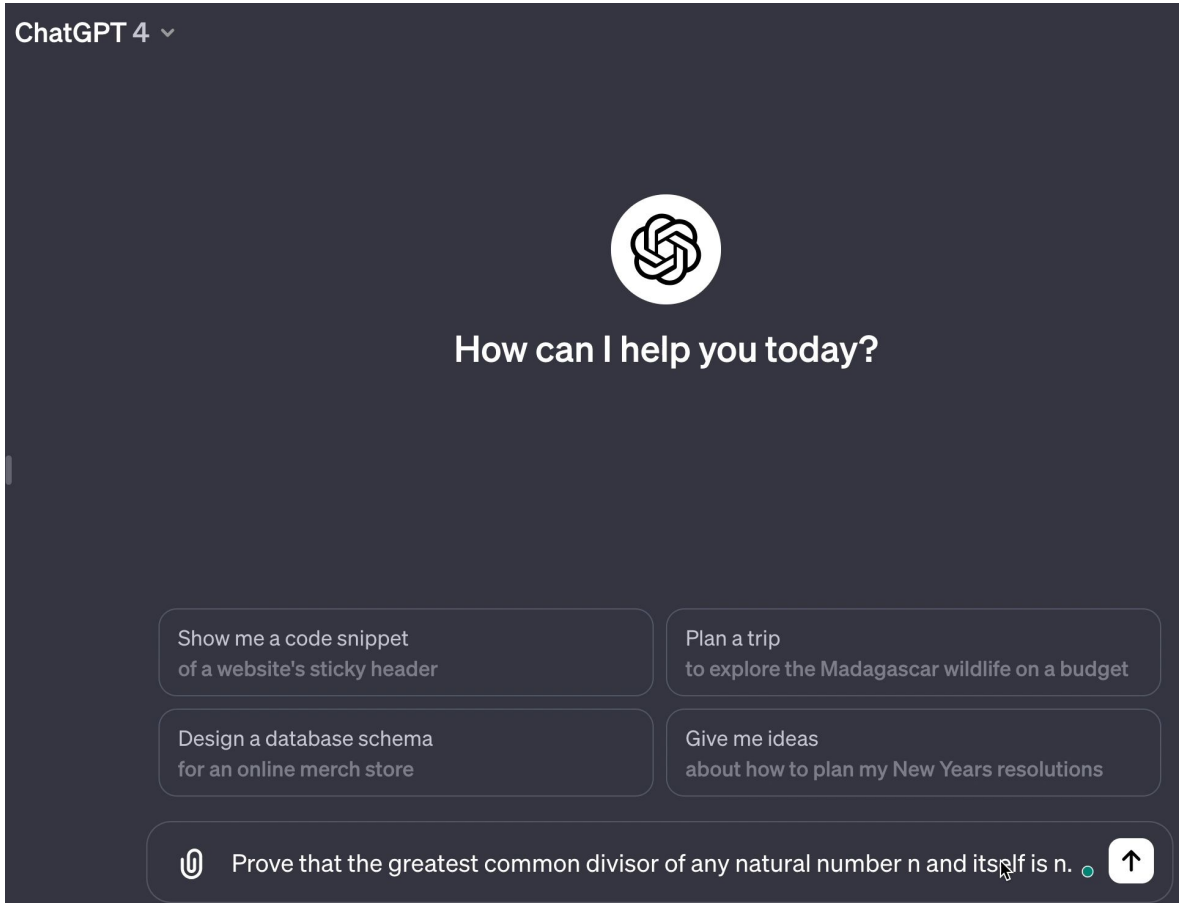



Wait.. but why should we care?

I.e. Why should we bother doing “theorem proving” for math & code? Why to enter the formal world?

- Why should we prove math theorems? -> Why should we ***formally*** prove math theorems?
- Why should we prove code properties? -> Why should we ***formally*** prove code properties?

Informal vs. Formal Mathematical Reasoning




```
theorem gcd_self (n : ℕ) : gcd n n = n
```

```
cases n
· unfold gcd
  rfl
· unfold gcd
  rw [mod_self]
  unfold gcd
  rfl
```

✓

```
cases n
· simp [gcd]
· rw [mod_self]
  unfold gcd
  rfl
```

✗

Important for LLMs to tackle advanced mathematics

- Grounded in environments that can provide feedback
- Simple and rigorous evaluation: formal proofs can be checked (no hallucination)

Informal ⟷ **Formal**

Checking Mathematical Proofs is Hard for Humans

Titans of Mathematics Clash Over Epic Proof of ABC Conjecture

Two mathematicians have found what they say is a hole at the heart of a proof that has convulsed the mathematics community for nearly six years.

Code Generation with LLMs

```
def gcd (x : int, y : int) -> int:
    """Compute the greatest common divisor of ``x`` and ``y``.
    >>> gcd(10, 5)
    5
    >>> gcd(2, 3)
    1
    >>> gcd(8, 12)
    4
    """
    if x == 0:
        return y
    if y == 0:
        return x
    if x < y:
        return gcd(x, y % x)
    return gcd(x % y, y)
```

Passing a few testing examples ≠ correctness

Code Generation with LLMs

```
In [3]: gcd(-10, -5)
```

What if x and y are negative?

```python
def gcd (x : int, y : int) -> int:
    """Compute the greatest common divisor of ``x`` and ``y``.
    >>> gcd(10, 5)
    5
    >>> gcd(2, 3)
    1
    >>> gcd(8, 12)
    4
    """
    if x == 0:
        return y
    if y == 0:
        return x
    if x < y:
        return gcd(x, y % x)
    return gcd(x % y, y)
```

Passing a few testing examples ≠ correctness

Code Generation with LLMs

```python
def gcd (x : int, y : int) -> int:
    """Compute the greatest common divisor of ``x`` and ``y``.
    >>> gcd(10, 5)
    5
    >>> gcd(2, 3)
    1
    >>> gcd(8, 12)
    4
    """
    if x == 0:
        return y
    if y == 0:
        return x
    if x < y:
        return gcd(x, y % x)
    return gcd(x % y, y)
```

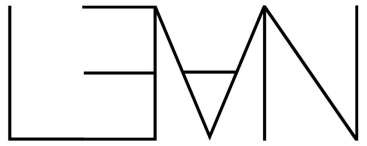
GitHub Copilot

```
In [3]: gcd(-10, -5)
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
Cell In[3], line 1
----> 1 gcd(-10, -5)

File ~/LeanDojo/tmp.py:16, in gcd(x, y)
     14     return x
     15 if x < y:
---> 16     return gcd(x, y % x)
     17 return gcd(x % y, y)

File ~/LeanDojo/tmp.py:16, in gcd(x, y)
     14     return x
     15 if x < y:
---> 16     return gcd(x, y % x)
     17 return gcd(x % y, y)

    [... skipping similar frames: gcd at line 16 (2981 times)]

File ~/LeanDojo/tmp.py:16, in gcd(x, y)
     14     return x
     15 if x < y:
---> 16     return gcd(x, y % x)
     17 return gcd(x % y, y)

File ~/LeanDojo/tmp.py:11, in gcd(x, y)
      2 def gcd (x : int, y : int) -> int:
      3     """Compute the greatest common divisor of ``x`` and ``y``.
      4     >>> gcd(10, 5)
      5     5
   (...)
      9     4
     10     """
---> 11     if x == 0:
     12         return y
     13     if y == 0:

RecursionError: maximum recursion depth exceeded in comparison
```

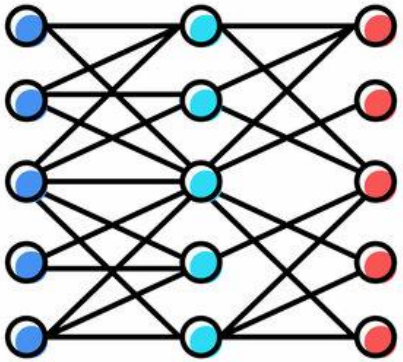
Passing a few testing examples ≠ correctness

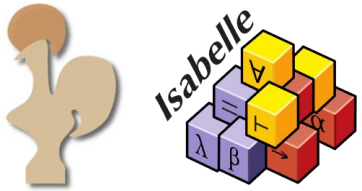
Automated Reasoning and Formal Proofs

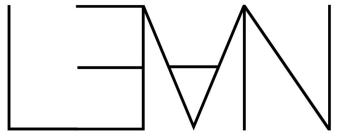
- **Interactive theorem proving**
  - Proof assistants such as Coq, Isabelle, and Lean
  - Expressive logic, e.g., dependent type theory
  - Successfully used in large formalization projects
  - **Lots of efforts from humans to write proofs**
  - **Proof automation is critical for wider adoption**
- **Automated theorem proving**
  - SMT solvers, model checkers, ATP systems in first-order logic, etc.
  - Minimal efforts from humans
  - Limited expressiveness
  - Difficult to scale

Theorem Provers + LLMs = Neuro-Symbolic







Machine learning ⟷ Proof assistants

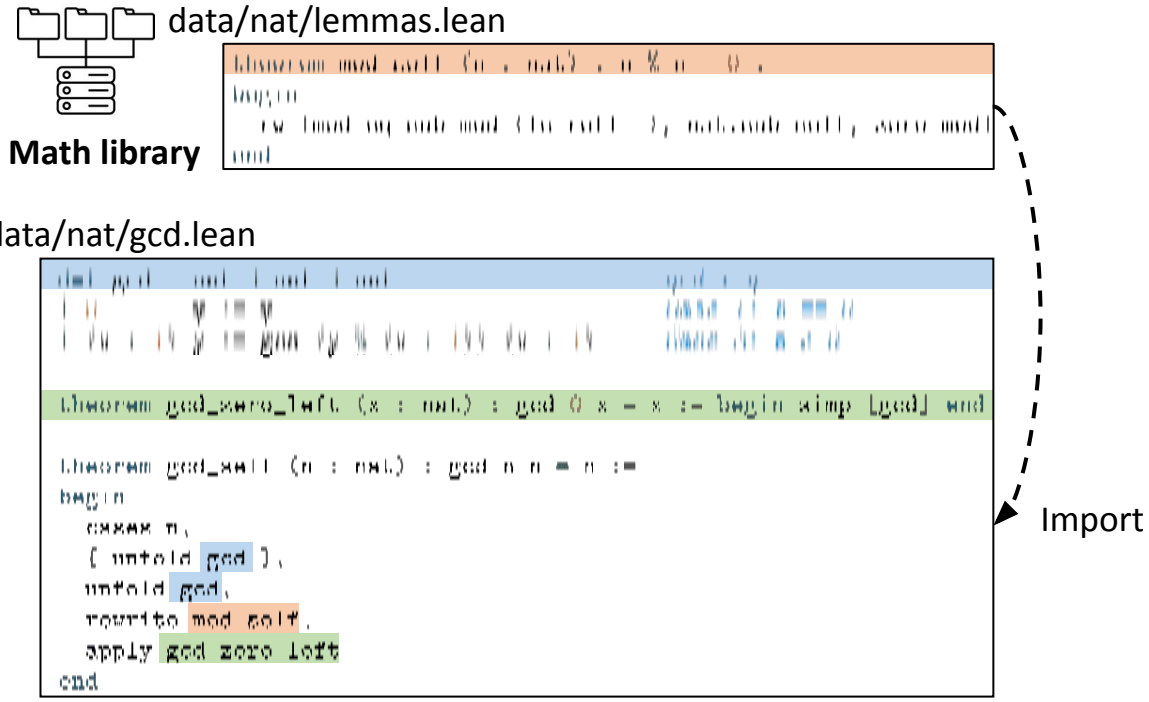
[NeurIPS 23] LeanDojo: Retrieval-Augmented LM + Lean

LeanDojo: Theorem Proving with Retrieval-Augmented Language Models

Kaiyu Yang[1], **Aidan M. Swope**[2], **Alex Gu**[3], **Rahul Chalamala**[1], **Peiyang Song**[4], **Shixing Yu**[5], **Saad Godil**[*], **Ryan Prenger**[2], **Anima Anandkumar**[1,2]
[1]Caltech, [2]NVIDIA, [3]MIT, [4]UC Santa Barbara, [5]UT Austin
https://leandojo.org

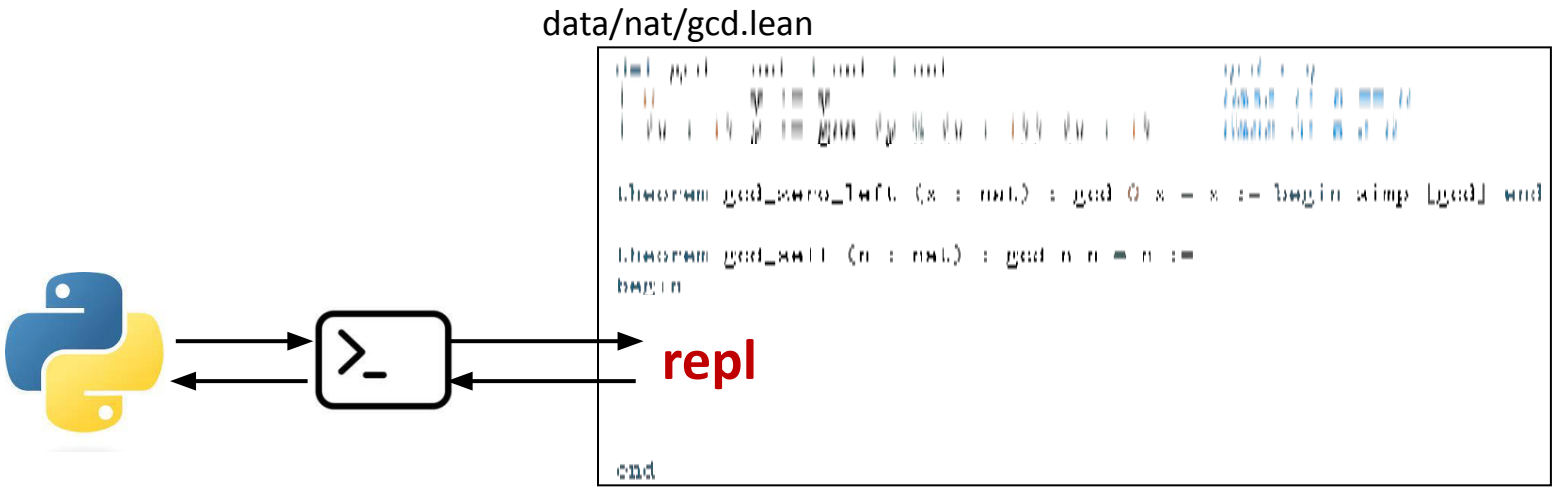
Data Extraction in LeanDojo

- ASTs, tactics
  - From Lean's parser
- Proof goals
  - From Lean's InfoTree
- Premises
  - Definitions, lemmas, etc.
  - Where they are used/defined
  - Also in the InfoTree



Programmatical Interaction in LeanDojo

- Replace the human-written proof with a single repl tactic
- repl performs IO to provide a command line interface for interacting with Lean
- Wrap the interface in any language, e.g., Python



ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises

State

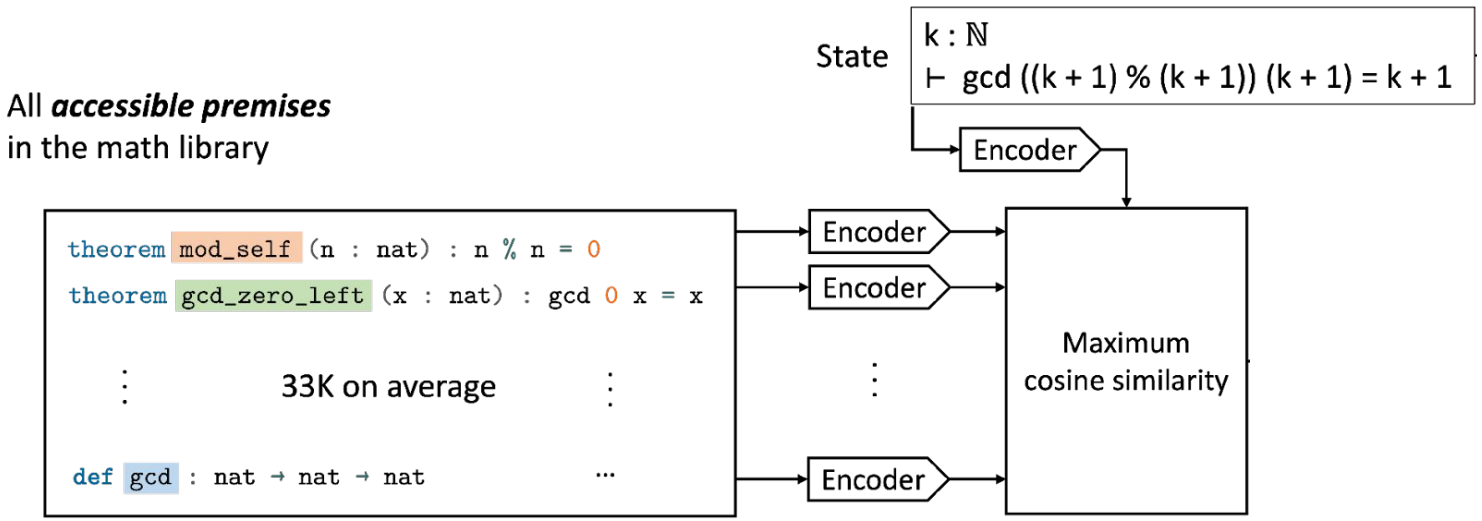
```
k : ℕ
⊢ gcd ((k + 1) % (k + 1)) (k + 1) = k + 1
```

All ***accessible premises***
in the math library

```
theorem mod_self (n : nat) : n % n = 0
theorem gcd_zero_left (x : nat) : gcd 0 x = x
⋮                 33K on average                 ⋮
def gcd : nat → nat → nat                        ⋯
```

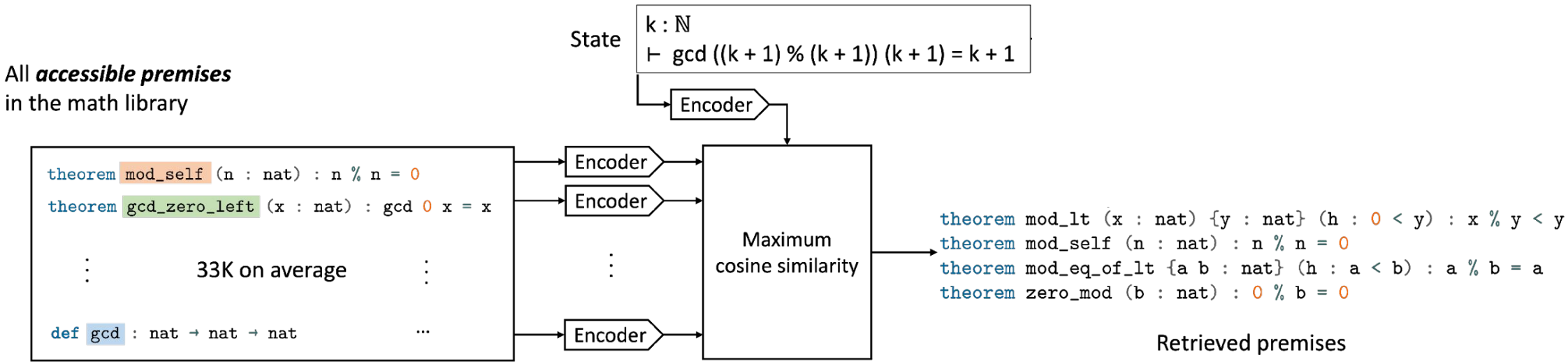
ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises



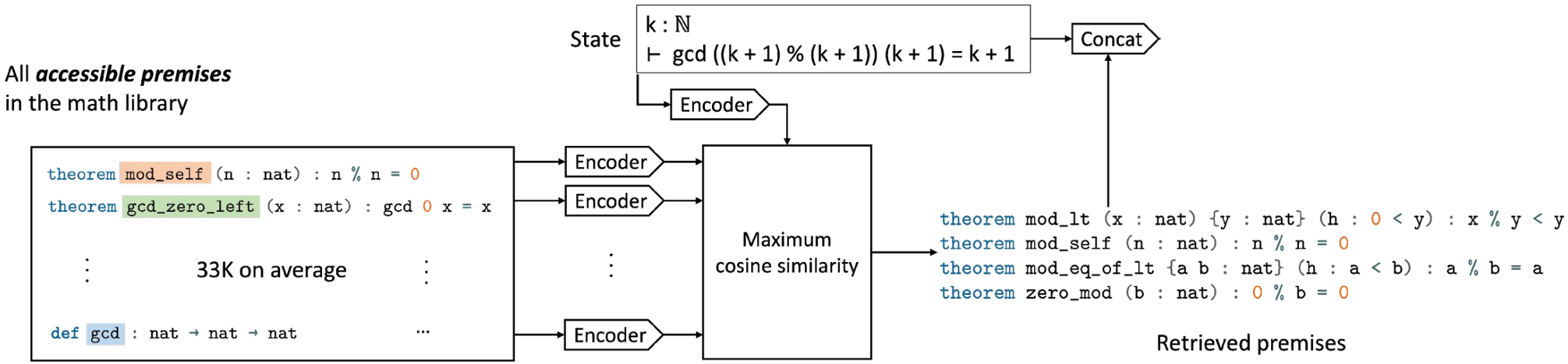
ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises



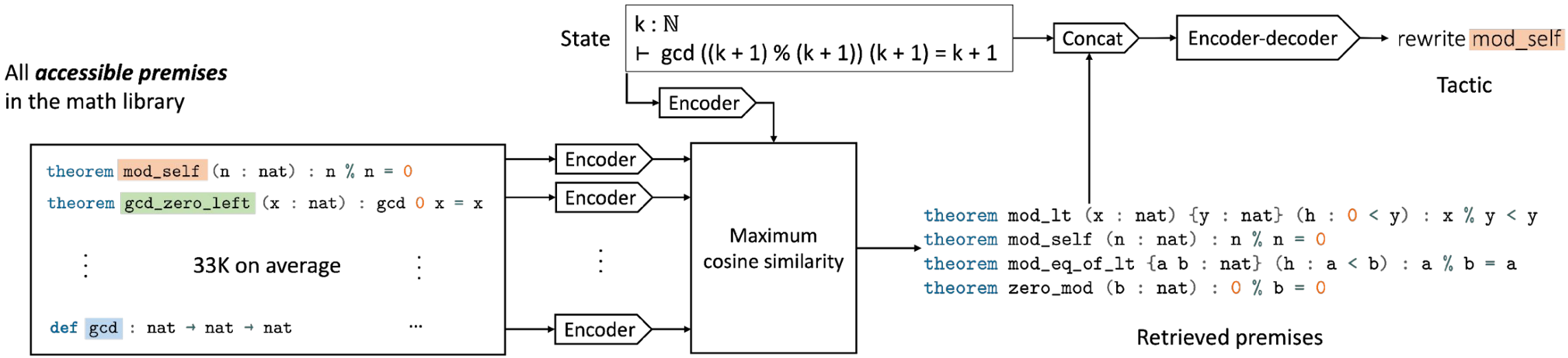
ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises
- Retrieved premises are concatenated with the state and used for **tactic generation**

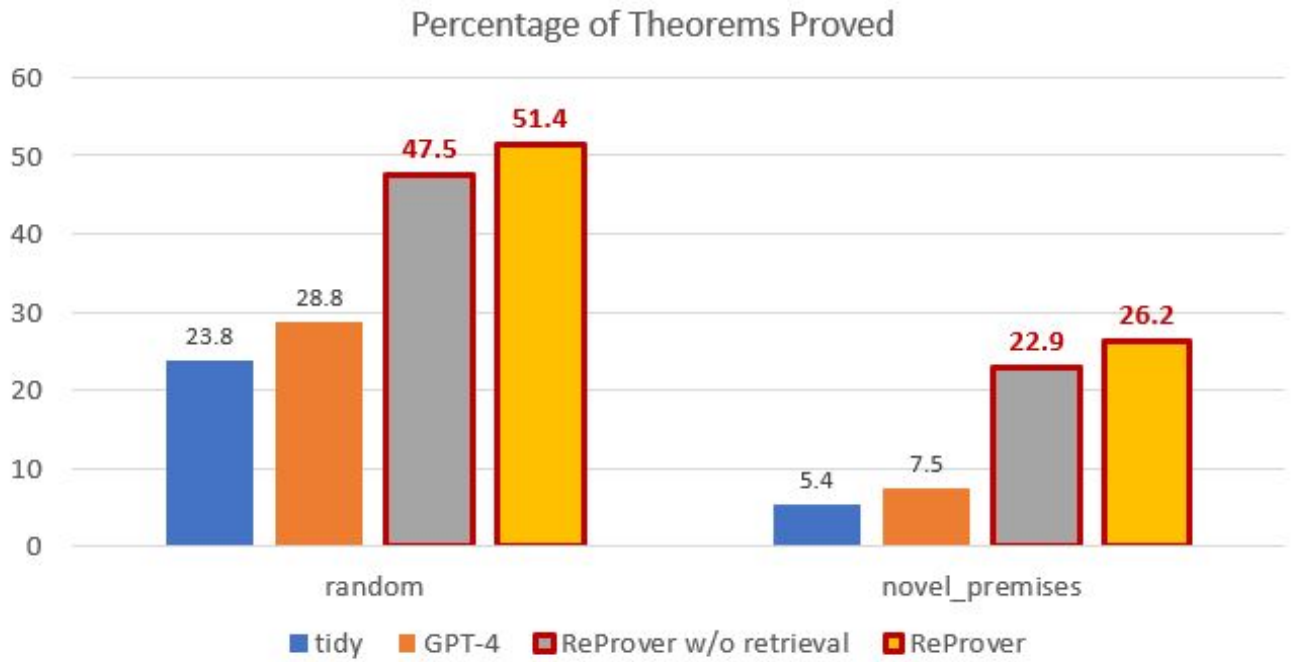


ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises
- Retrieved premises are concatenated with the state and used for **tactic generation**

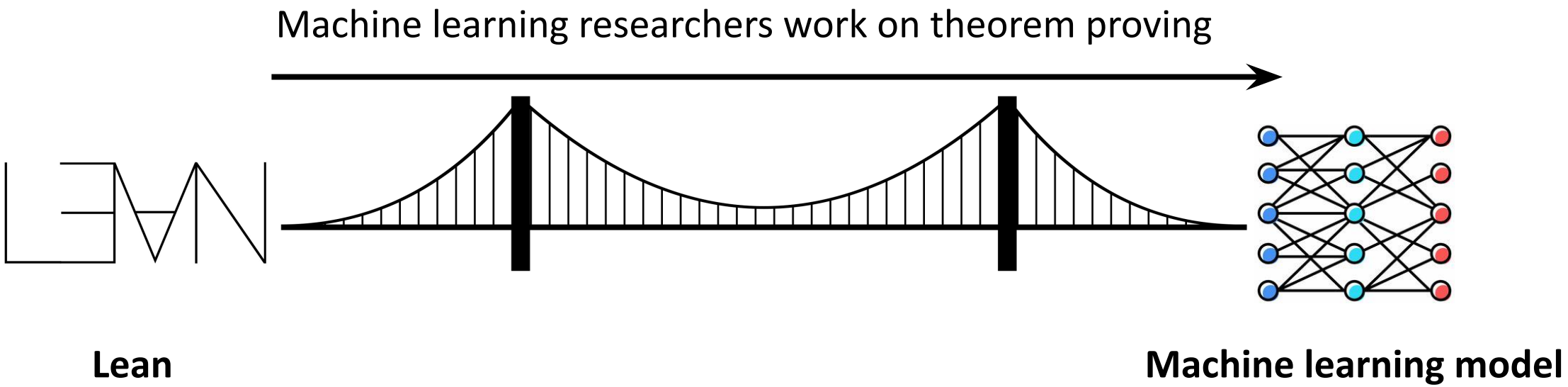


ReProver: Retrieval-Augmented Prover

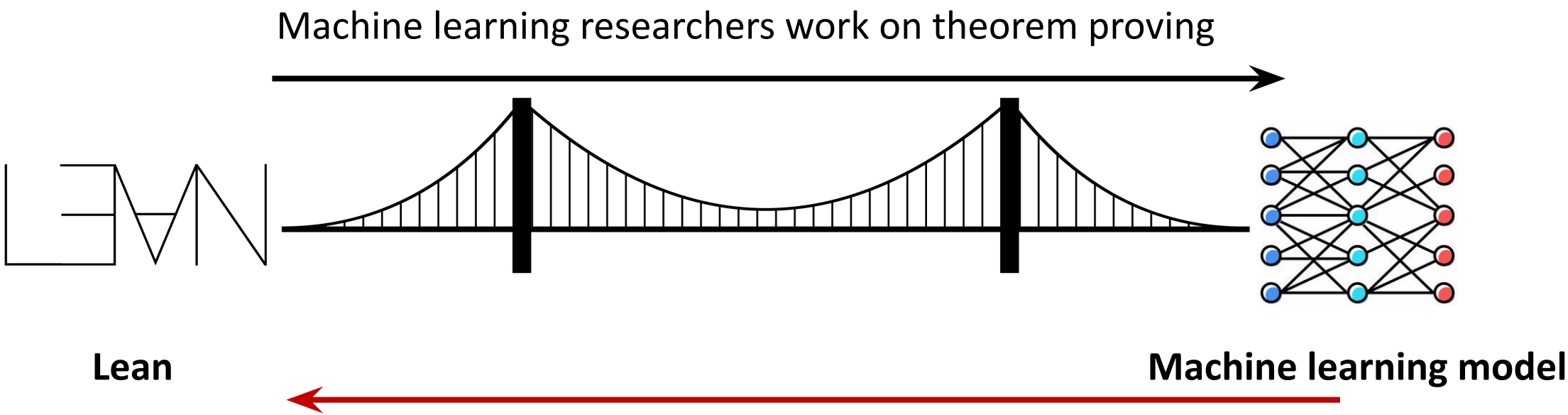


Method	random	novel_premises
tidy	23.8	5.3
GPT-4	29.0	7.4
ReProver (ours)	**51.2**	**26.3**
w/o retrieval	47.6	23.2

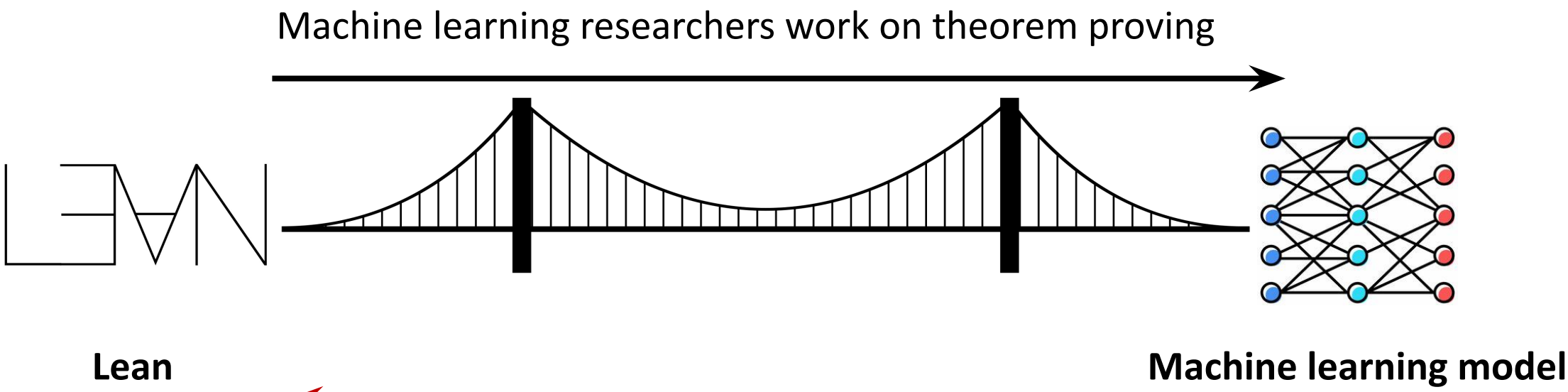
Bridging Machine Learning and Theorem Proving



Bridging Machine Learning and Theorem Proving



Bridging Machine Learning and Theorem Proving



- Run on CPUs reasonably fast
- Integrated into VSCode
- Care about a specific domain, not aggregated performance on mathlib

[NeuS 25] Lean Copilot: 1.2k+ stars, 2nd among all

Lean Copilot: Large Language Models as Copilots for Theorem Proving in Lean

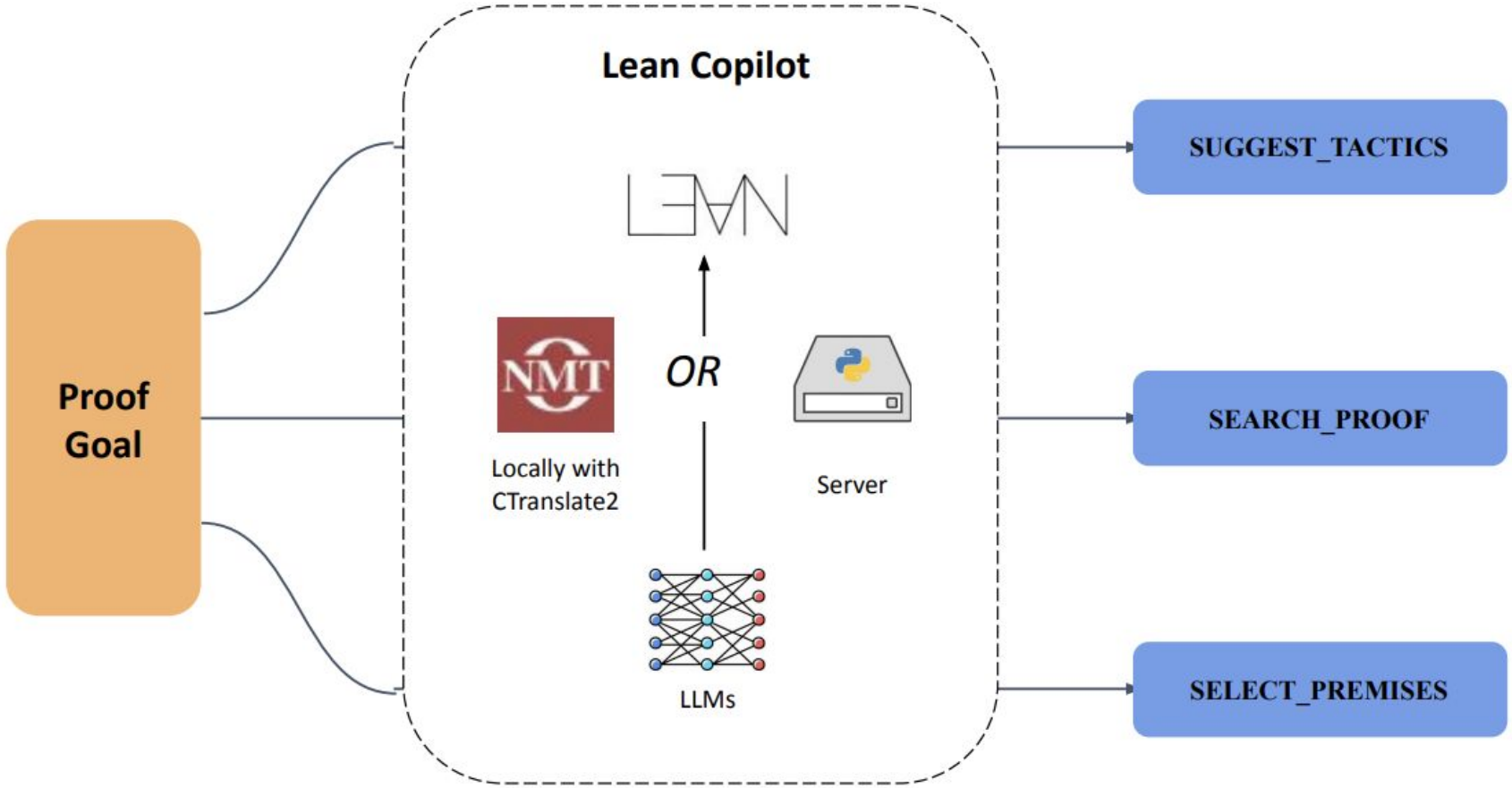
Peiyang Song PSONG@CALTECH.EDU
California Institute of Technology, Pasadena, CA, U.S.A.

Kaiyu Yang KAIYUY@CALTECH.EDU
California Institute of Technology, Pasadena, CA, U.S.A.

Anima Anandkumar ANIMA@CALTECH.EDU
California Institute of Technology, Pasadena, CA, U.S.A.

Lean Copilot Toolkit

Easy to install just like any Lean package
Run locally on most laptops w/o GPUs
Respond in seconds



Tactic Suggestion

```
import LeanCopilot


theorem add_abc (a b c : Nat) : a + b + c = a + c + b := by
  suggest_tactics
```

▼ Tactic state

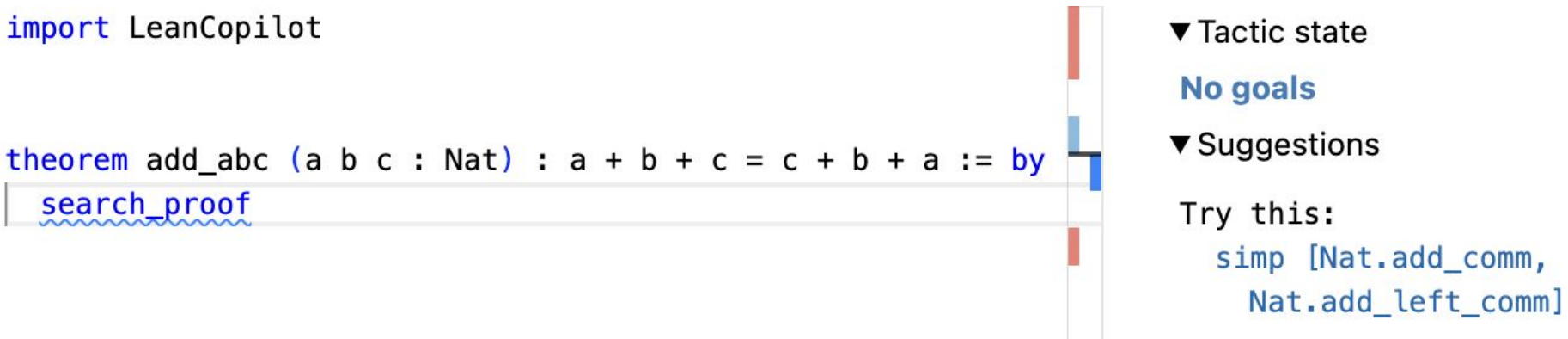
No goals

▼ Suggestions

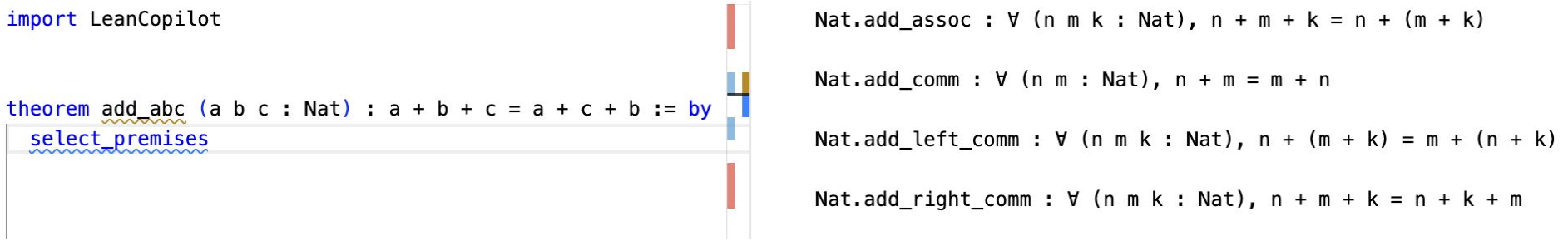
Try these:

- `apply Nat.add_right_comm`
- `rw [Nat.add_assoc ]`
  Remaining subgoals:
  ⊢ a + (b + c) = a + c + b
- `rw [Nat.add_comm ]`
  Remaining subgoals:
  ⊢ c + (a + b) = a + c + b
- `simp [Nat.add_assoc ]`
  Remaining subgoals:
  ⊢ a + (b + c) = a + (c + b)

Proof Search



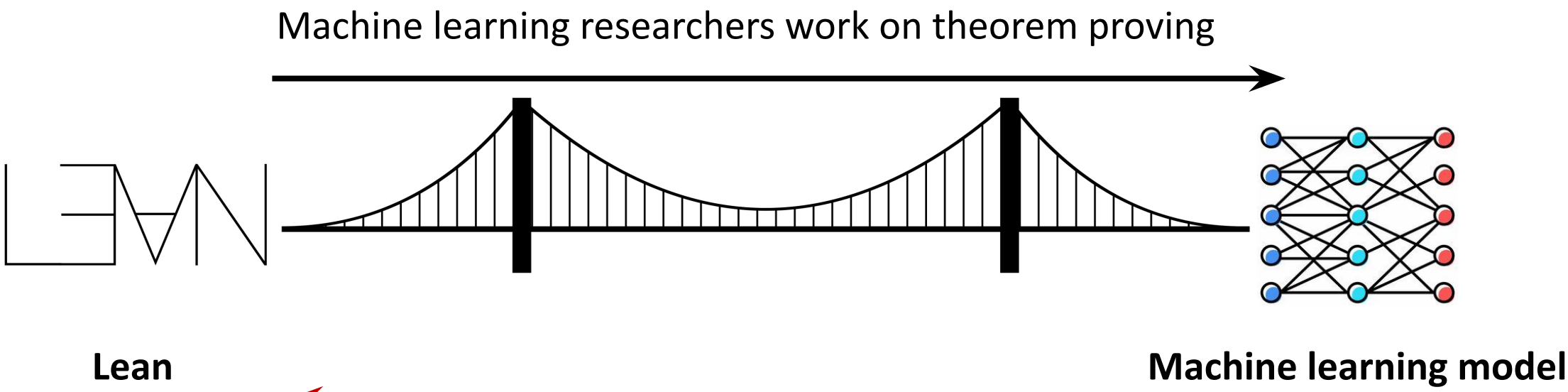
Premise Selection



With rich annotations!

- In-scope premises: provide type information and doc strings
- Out-of-scope premises: provide complete definition + instruction on usage

Bridging Machine Learning and Theorem Proving



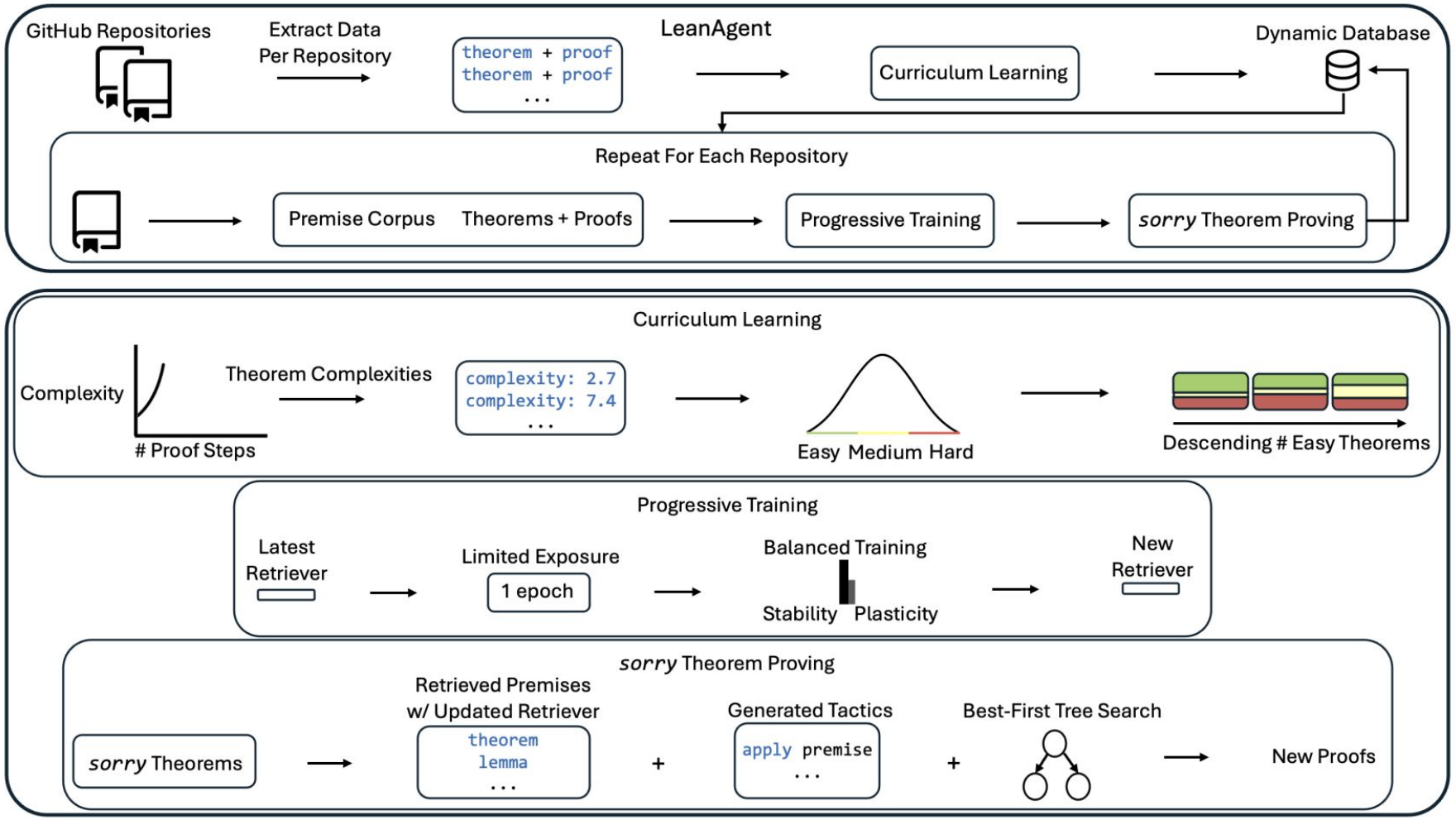
- Run on CPUs reasonably fast
- Integrated into VSCode
- Care about a specific domain, not aggregated performance on mathlib

[ICLR 25] LeanAgent: Lifelong Learning + TP

LeanAgent: Lifelong Learning for Formal Theorem Proving

Adarsh Kumarappan[*,1], **Mo Tiwari**[*,2], **Peiyang Song**[1], **Robert Joseph George**[1], **Chaowei Xiao**[3], **Anima Anandkumar**[1]
[1]California Institute of Technology, [2]Stanford University, [3]University of Wisconsin, Madison
{adarsh, psong, rgeorge, anima}@caltech.edu, motiwari@stanford.edu, cxiao34@wisc.edu

[ICLR 25] LeanAgent: Lifelong Learning + TP



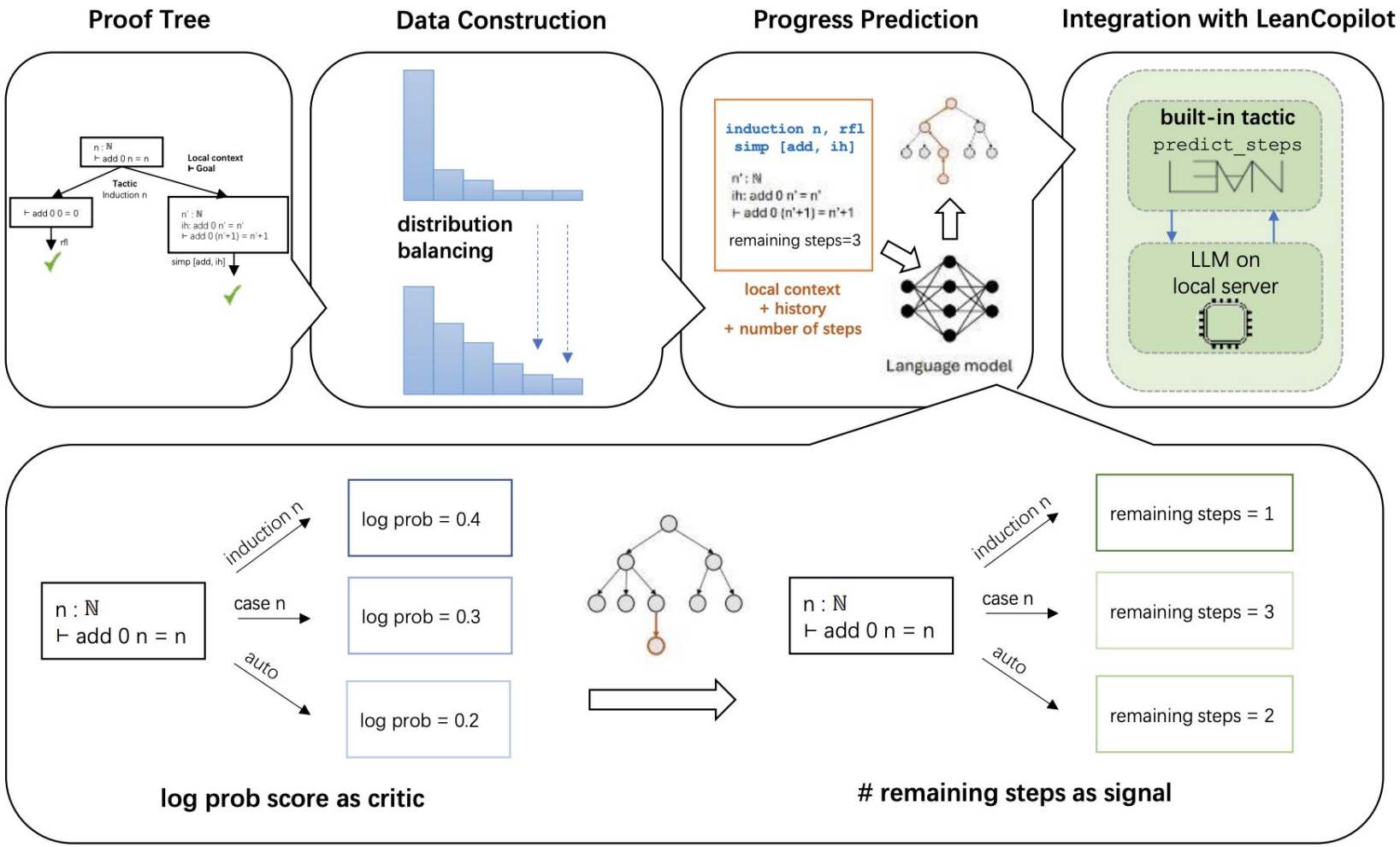
[ICML 25 VerifAI] LeanProgress: Predict # Steps

LeanProgress: Guiding Search for Neural Theorem Proving via Proof Progress Prediction

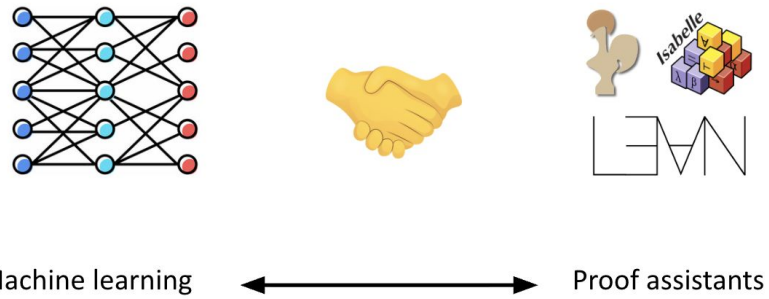
---

Suozhi Huang [1] **Peiyang Song** [2] **Robert Joseph George** [2] **Anima Anandkumar** [2]

[ICML 25 VerifAI] LeanProgress: Predict # Steps



More along this line: ML + TP



I.e., what am I working on these days?

- **Towards mathematically important problems other than TP**:
  - E.g., automated conjecturing + interestingness-guided evaluation (out soon!)
- **Towards more practically useful, research-level math**:
  - Agentic approaches for theorem proving
  - Auto-formalization & auto-informalization
- **Towards extending TP beyond formal math**:
  - Verifiable code generation -> repo-level, realistic scenarios (out soon!)
  - Development of CSLib + tools for automation

Research Interest

Keywords: **LLM reasoning**, **agentic AI**, **neuro-symbolic AI**.

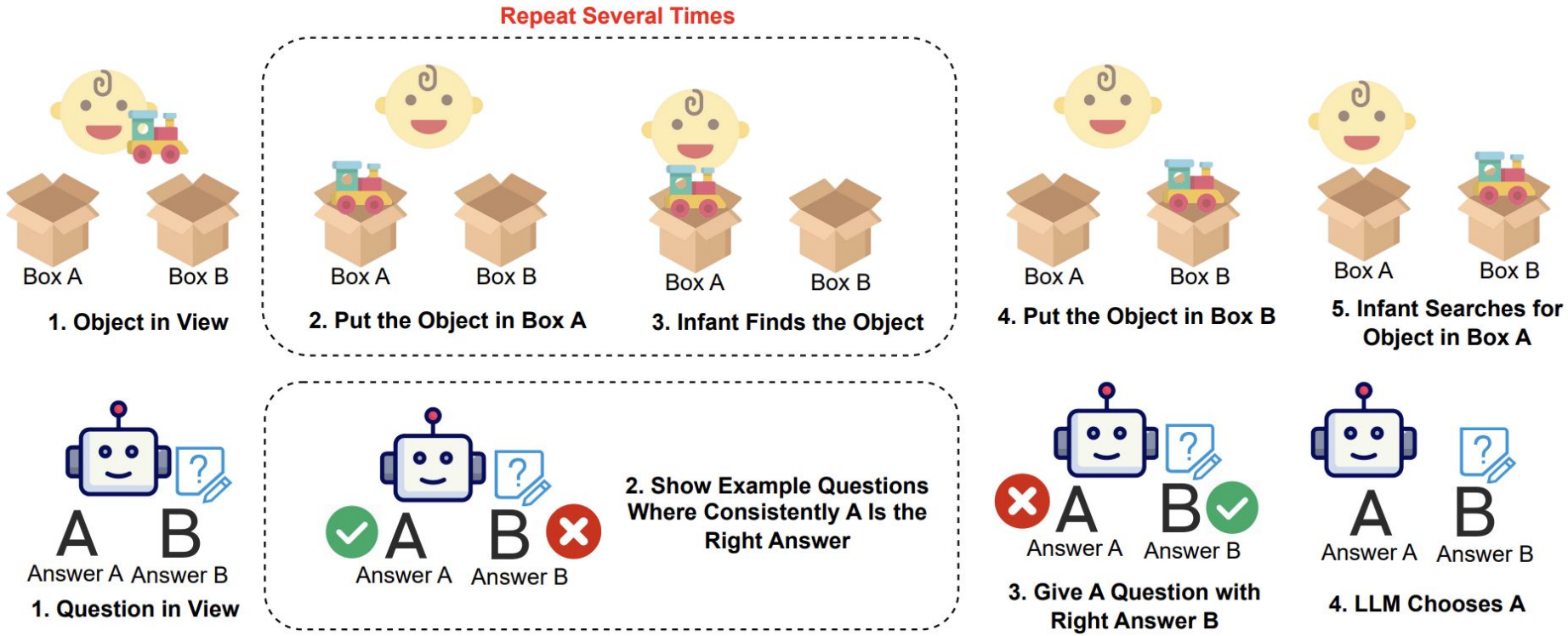
- One **central axis** (**AI for Math & Code**): Integrating LLM reasoning, agentic AI, and neuro-symbolic methods by combining *neural* models (LLMs) with formal *symbolic* systems (such as *Lean*) to advance LLM-based agents for formal reasoning in *math* and *code*.
- Broader **LLM Reasoning**: Extending LLM reasoning beyond formal domains, exploring how LLMs can tackle *informal* reasoning in natural language, inspired by *cognitive science* principles and studies of *human-like* reasoning.
- Broader **Neuro-Symbolic AI**: Building general *neuro-symbolic* approaches towards *fundamental* AI systems beyond formal reasoning, such as developing *energy-efficient inference* and *machine translation* across idioms and languages.

Broader LLM Reasoning

In-Context Learning May Not Elicit Trustworthy Reasoning: A-Not-B Errors in Pretrained Language Models

Pengrui Han[*†1,3], **Peiyang Song**[*2], **Haofei Yu**[3], **Jiaxuan You**[3]
[1]Carleton College, [2]California Institute of Technology
[3]University of Illinois Urbana-Champaign
barryhan@carleton.edu, psong@caltech.edu

- [EMNLP 24] ICL Does Not Always Elicit Trustworthy Reasoning

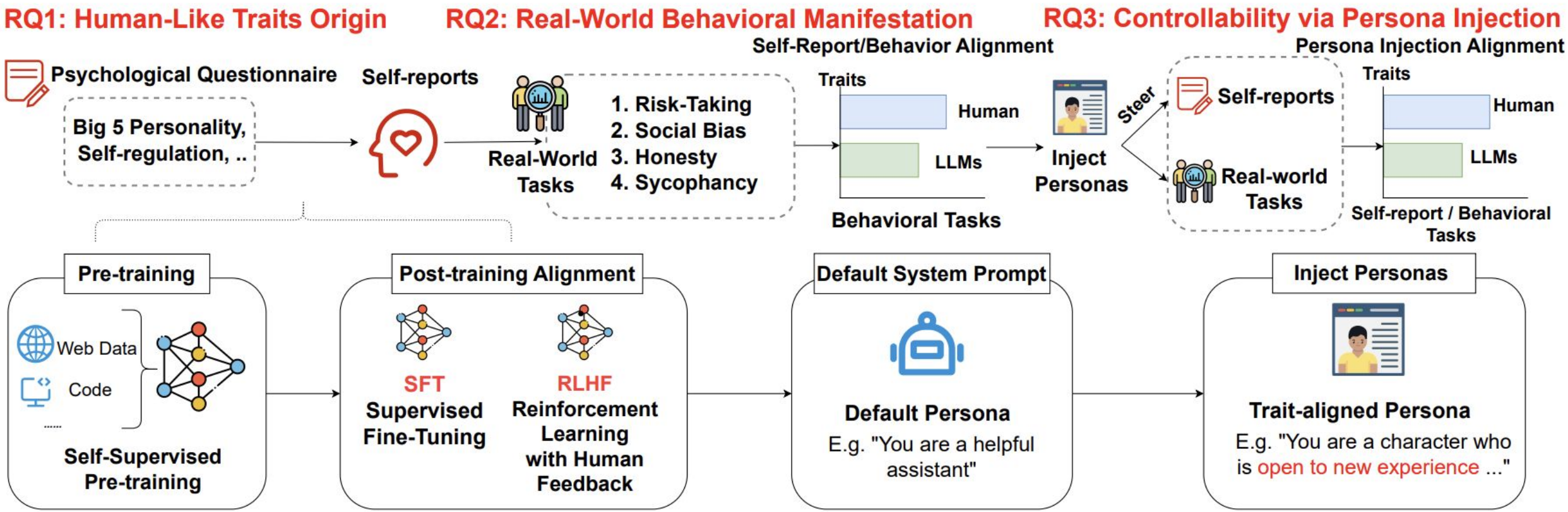


Broader LLM Reasoning

THE PERSONALITY ILLUSION: REVEALING DISSOCIATION BETWEEN SELF-REPORTS & BEHAVIOR IN LLMS

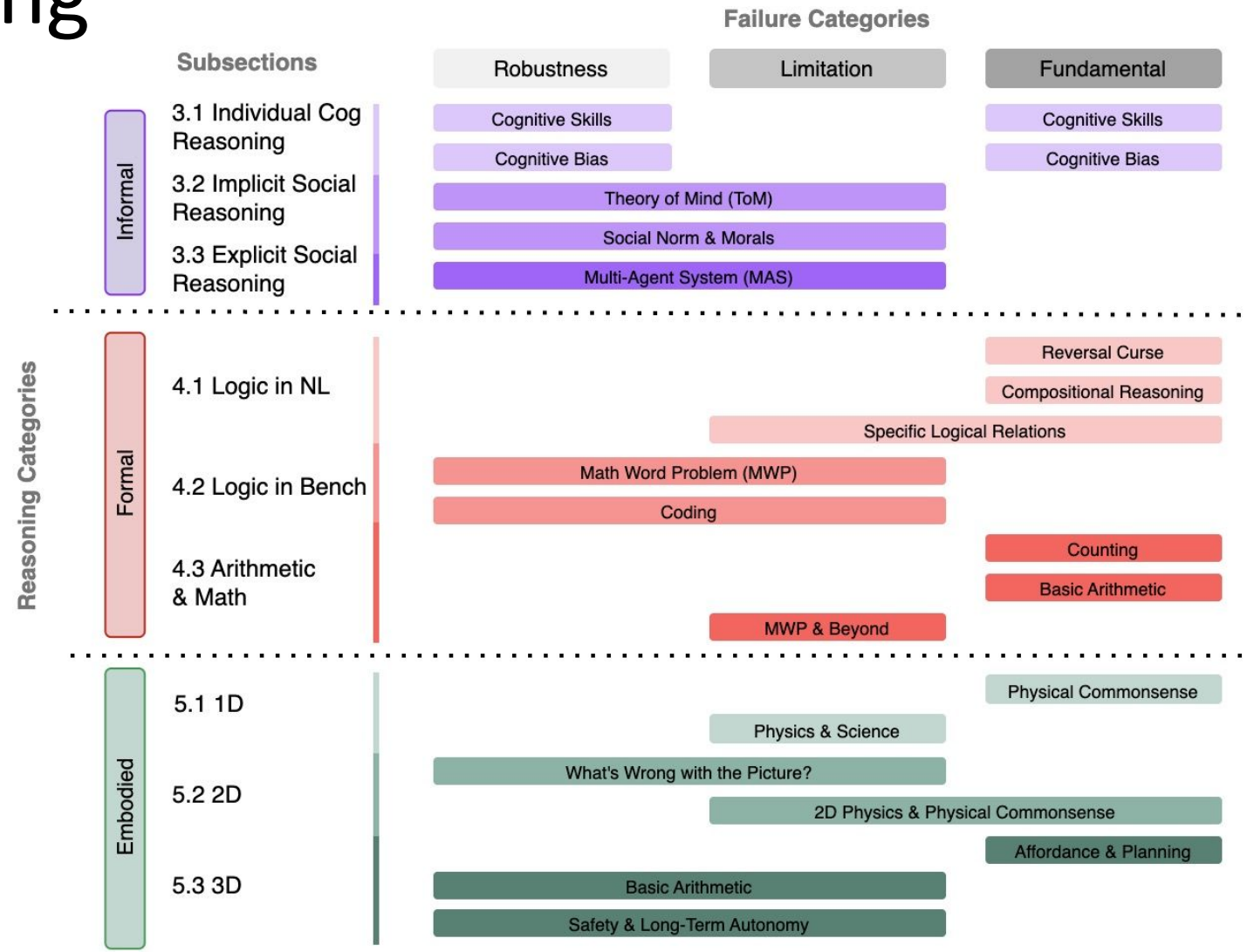
Pengrui Han[*1,2] **Rafal Kocielnik**[*1] **Peiyang Song**[1] **Ramit Debnath**[3] **Dean Mobbs**[1]
Anima Anandkumar[1] **R. Michael Alvarez**[1]
[1]Caltech [2]UIUC [3]University of Cambridge
phan12@illinois.edu, rafalko@caltech.edu
https://psychology-of-ai.github.io/

- [NeurIPS 25 LAW Workshop] Personality Illusion: Self-Report ≠ Real-World Behavior



Broader LLM Reasoning

- [ICML 25 AI4MATH Workshop] Survey on LLM Reasoning Failures
- Unify the field – structured failure studies can identify common patterns to improve on



A Survey on Large Language Model Reasoning Failures

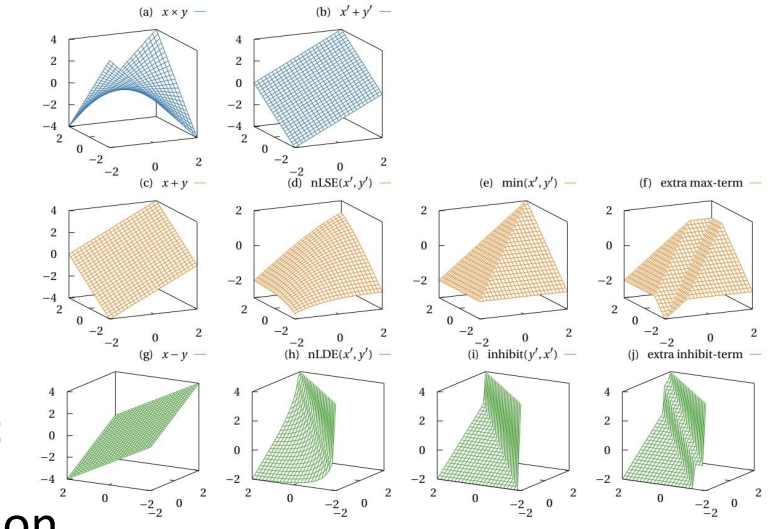
Peiyang Song [*1,2] Pengrui Han [*3] Noah Goodman [1]

Research Interest

Keywords: **LLM reasoning**, **agentic AI**, **neuro-symbolic AI**.

- One **central axis** (**AI for Math & Code**): Integrating LLM reasoning, agentic AI, and neuro-symbolic methods by combining *neural* models (LLMs) with formal *symbolic* systems (such as *Lean*) to advance LLM-based agents for formal reasoning in *math* and *code*.
- Broader **LLM Reasoning**: Extending LLM reasoning beyond formal domains, exploring how LLMs can tackle *informal* reasoning in natural language, inspired by *cognitive science* principles and studies of *human-like* reasoning.
- Broader **Neuro-Symbolic AI**: Building general *neuro-symbolic* approaches towards *fundamental* AI systems beyond formal reasoning, such as developing *energy-efficient inference* and *machine translation* across idioms and languages.

Broader Neuro-Symbolic AI: MLSys

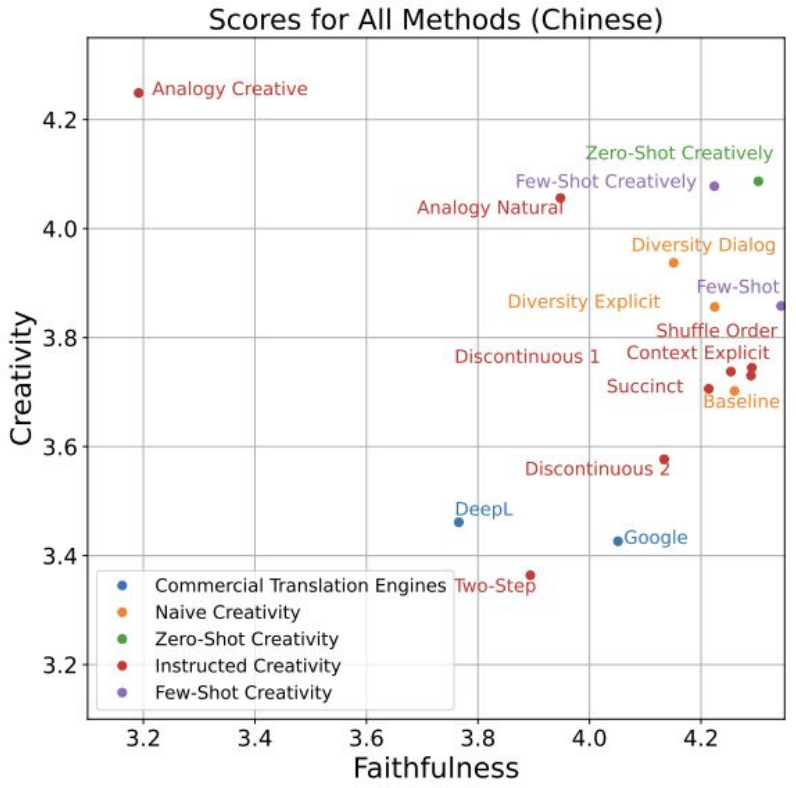


- **[ASPLOS 24] Energy Efficient Convolutions with Temporal Arithmetic**
  - Using a new temporal arithmetic with a negative log transformation
  - Improved the energy per pixel of each convolution frame by more than **2×** compared to the state-of-the-art; improved the energy delay product by **four orders of magnitude**

- **[IEEE Micro 25 Top Pick 🏆] Delay Space Arithmetic and Architecture**
  - Negative-logarithmic delay space arithmetic as a completely new approach to temporal coding
  - Under this approach, general purpose arithmetic is transformed to a “soft” version of the standard temporal operations in such a way that **preserves all of the algebraic identities**

Broader Neuro-Symbolic AI: Machine Translation

- [EMNLP 24] Creative and Context-Aware Translation of East Asian Idioms with GPT-4
  - Challenging because..
    - Low resource lang
    - Linguistically rich
  - Need to balance
    - Faithfulness
    - Creativity
    - Cultural meanings
  - New Pareto-optimality



1. Idiom
刮目相看

2. Sentences
小明的成绩提高得非常快，让老师和同学们都刮目相看。

3. Context-Aware Translations
Xiaoming's grades soared impressively, leaving both teachers and classmates in awe.

Extracted Spans
- leaving both teachers and classmates in awe
- taken everyone by surprise
- like a phoenix reborn from its ashes
- a blazing comet
- earned everyone's admiration
- with newfound respect
- ...

Human Reference (Sentences Not Available)
- treat somebody with increased respect
- look at somebody with new eyes
- have a completely new appraisal of somebody
- regard somebody with special esteem

Research Interest

Keywords: **LLM reasoning**, **agentic AI**, **neuro-symbolic AI**.

- One **central axis** (**AI for Math & Code**): Integrating LLM reasoning, agentic AI, and neuro-symbolic methods by combining *neural* models (LLMs) with formal *symbolic* systems (such as *Lean*) to advance LLM-based agents for formal reasoning in *math* and *code*.
- Broader **LLM Reasoning**: Extending LLM reasoning beyond formal domains, exploring how LLMs can tackle *informal* reasoning in natural language, inspired by *cognitive science* principles and studies of *human-like* reasoning.
- Broader **Neuro-Symbolic AI**: Building general *neuro-symbolic* approaches towards *fundamental* AI systems beyond formal reasoning, such as developing *energy-efficient inference* and *machine translation* across idioms and languages.

LLM Reasoning for Math & Code

Happy to take questions!

Peiyang Song

California Institute of Technology

psong@caltech.edu