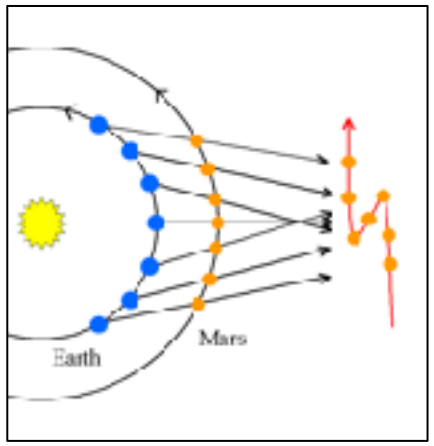


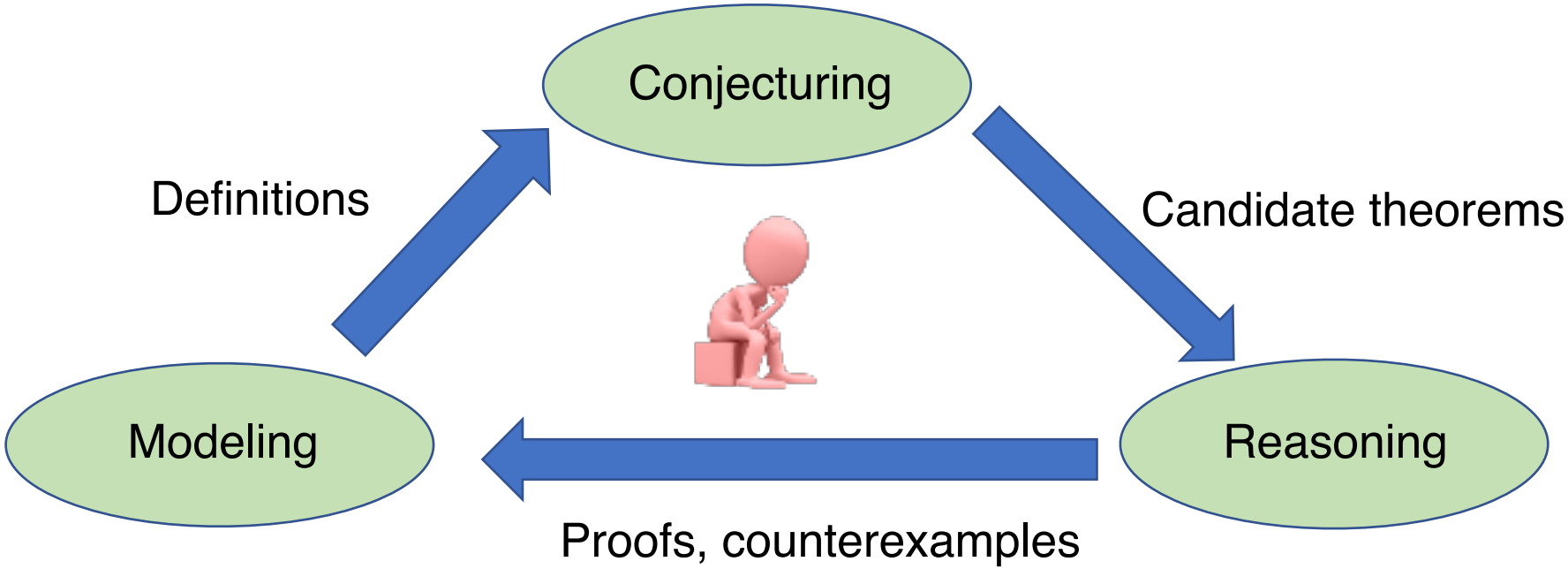
Swarat Chaudhuri
Computer Science
The University of Texas at Austin

TEXAS
The University of Texas at Austin

Trishul

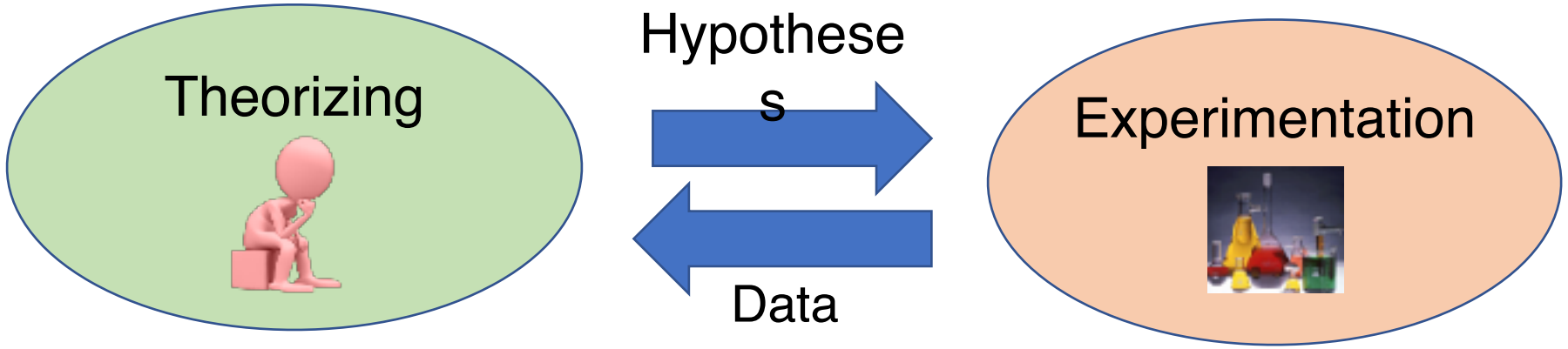
Abstraction and Discovery with Large Language Model Agents

Mathematical discovery



AI for math: Automate conjecturing and proof

Scientific discovery



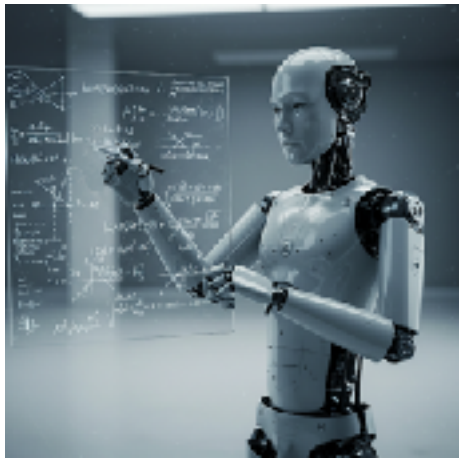
AI for science: Automate hypothesis generation and experiment design

Key ideas

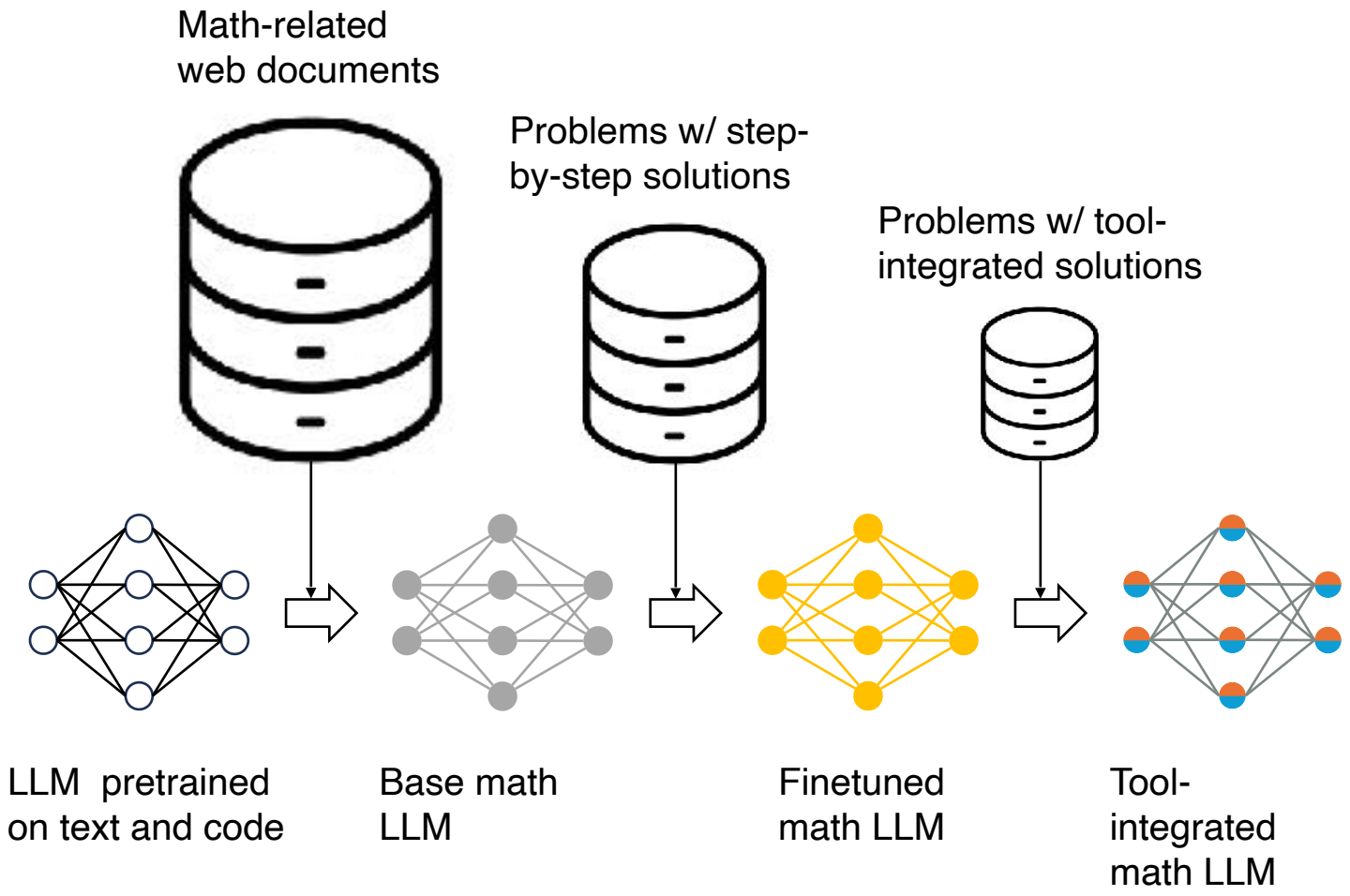
- Systematically **search** spaces of hypotheses, conjectures, and proofs
- Use **prior knowledge** to prioritize directions of search
- **Learn**, from experience, how and how not to search
- Discover **abstract** concepts and tools, speeding up both search and learning.

This talk: LLM agents with all four capabilities.

AI for Mathematical Discovery



Are neural networks all you need?



Problem: Suppose that the sum of the squares of two complex numbers x and y is 7, and the sum of their cubes is 10. List all possible values for $x + y$, separated by commas.

Solution: Let's use `sympy` to calculate and print all possible values for $x + y$.

```
def possible_values():
    x, y = symbols("x y")
    eq1 = Eq(x**2 + y**2, 7)
    eq2 = Eq(x**3 + y**3, 10)
    solutions = solve((eq1, eq2), (x, y))
    return [simplify(sol[0] + sol[1]) for sol in solutions]

print(possible_values())
```

>>> [-5, -5, 1, 1, 4, 4]

Removing duplicates, the possible values for $x + y$ are \boxed{-5, 1, 4}

Fleureau et al. "How NuminaMath Won the 1st AIMO Progress Prize" 2024

Are neural networks all you need?

Putnam 2024, Problem A2

A2 For which real polynomials p is there a real polynomial q such that

$$p(p(x)) - x = (p(x) - x)^2 q(x)$$

for all real x?

Response from OpenAI o1

Real Polynomial Solutions to Functional Equation

Thought for 5m 44s

Answer: The only solutions are linear polynomials. Specifically, a real polynomial p satisfies the given equation if and only if p is of one of the following forms:

1. $p(x) = x + c$ for any real c, and
   - If $c = 0$, then $p(x) = x$ and any polynomial q works.
   - If $c \neq 0$, then $q(x)$ must be the constant polynomial $q(x) = \frac{2}{c}$.
2. $p(x) = -x + b$ for any real b, and in this case, $q(x) = 0$.

No other polynomial p works.

Weaknesses of the neural-only approach

Data scarcity

- Need traces or reward functions that enable rigorous mathematical reasoning
- This is difficult beyond high-school or competition settings.

Lack of verifiability

- Natural-language reasoning is hard to verify
- In applications like system verification, edge cases are especially critical.

Open question: Will scaling solve these problems?

Alternative: Formal representations

Informal Problem Statement

"Prove that if a number is even, its square is even as well"

Neural autoformalizer

Formal Problem Statement

```
theorem mod_arith_2
(x : ℕ) : x % 2 = 0
    → (x * x) % 2 = 0
```

Formal Problem Statement

```
theorem mod_arith_2
(x : ℕ) : x % 2 = 0
    → (x * x) % 2 = 0
```

Neural prover

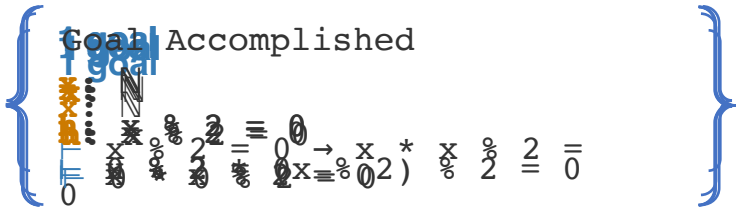
```
begin
  intro h,
  rw nat.mul_mod,
  rw h,
  rw nat.zero_mul,
  refl,
end
```

Formal proof assistant

QED!

Example: Formal Representations

```
theorem mod_arith_2
(x : ℕ) : x % 2 = 0 → (x * x) % 2 = 0 :=
begin
  intro h,
  rw nat.mul_mod,
  rw h,
  rw nat.zero_mul,
  refl,
end
```

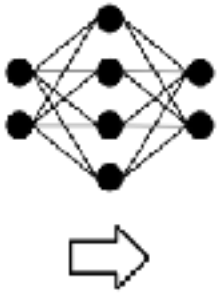


Autoformalization

Informal math

Theorem 1. *There exists an infinite number of primes.*

Proof. Let n be an arbitrary positive integer, and let $p \in \mathbb{Z}^+$ be a prime factor of $n!+1$. We can derive $p > n$ by noting that $n! + 1$ cannot be divided by positive integers from 2 to n. Since n is arbitrary, we have proved that the number of primes is infinite. □

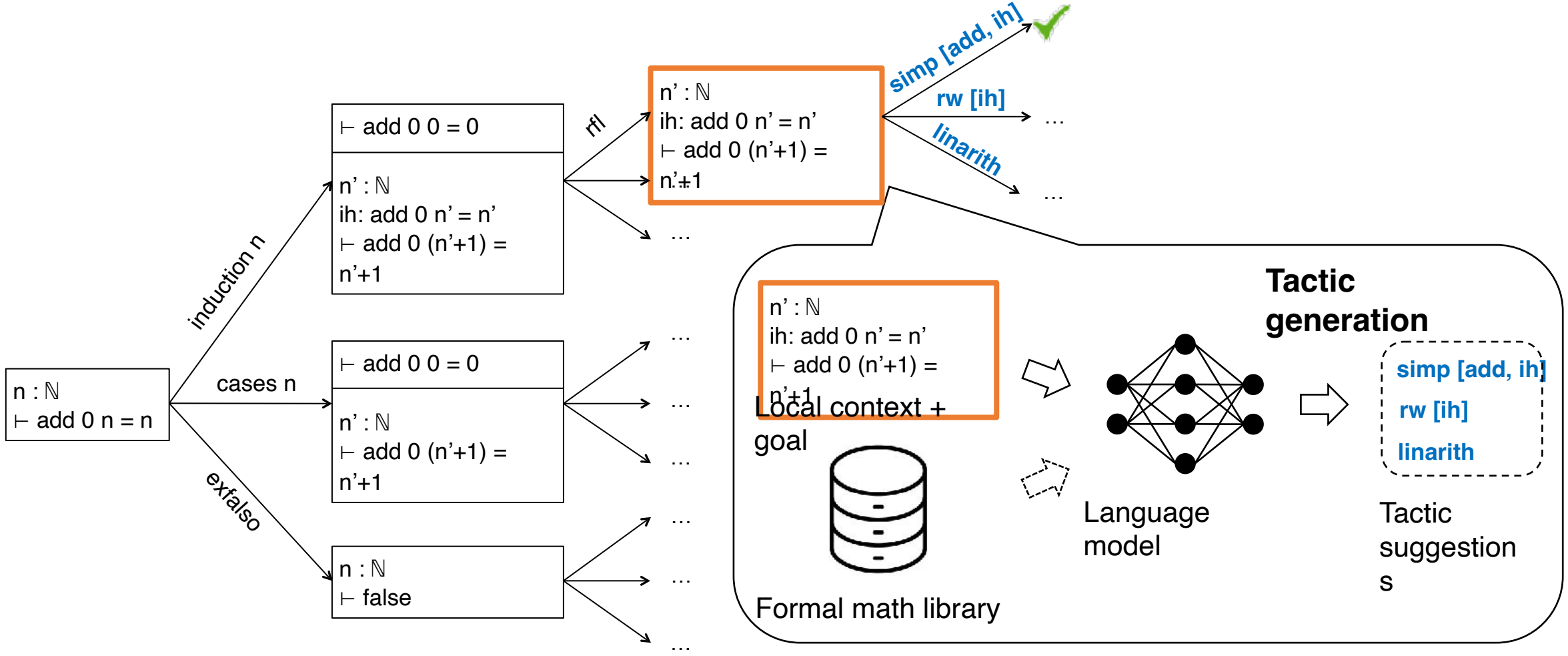


Formal theorem (and proof)

```
theorem exists_infinite_primes (n : ℕ) : ∃ p, n ≤ p ∧ Prime p :=
  let p := minFac (n ! + 1)
  have f1 : n ! + 1 ≠ 1 := ne_of_gt <| succ_lt_succ <| factorial_pos _
  have pp : Prime p := minFac_prime f1
  have np : n ≤ p :=
    le_of_not_ge fun h =>
      have h₁ : p ∣ n ! := dvd_factorial (minFac_pos _) h
      have h₂ : p ∣ 1 := (Nat.dvd_add_iff_right h₁).2 (minFac_dvd _)
      pp.not_dvd_one h₂
  ⟨p, np, pp⟩
```

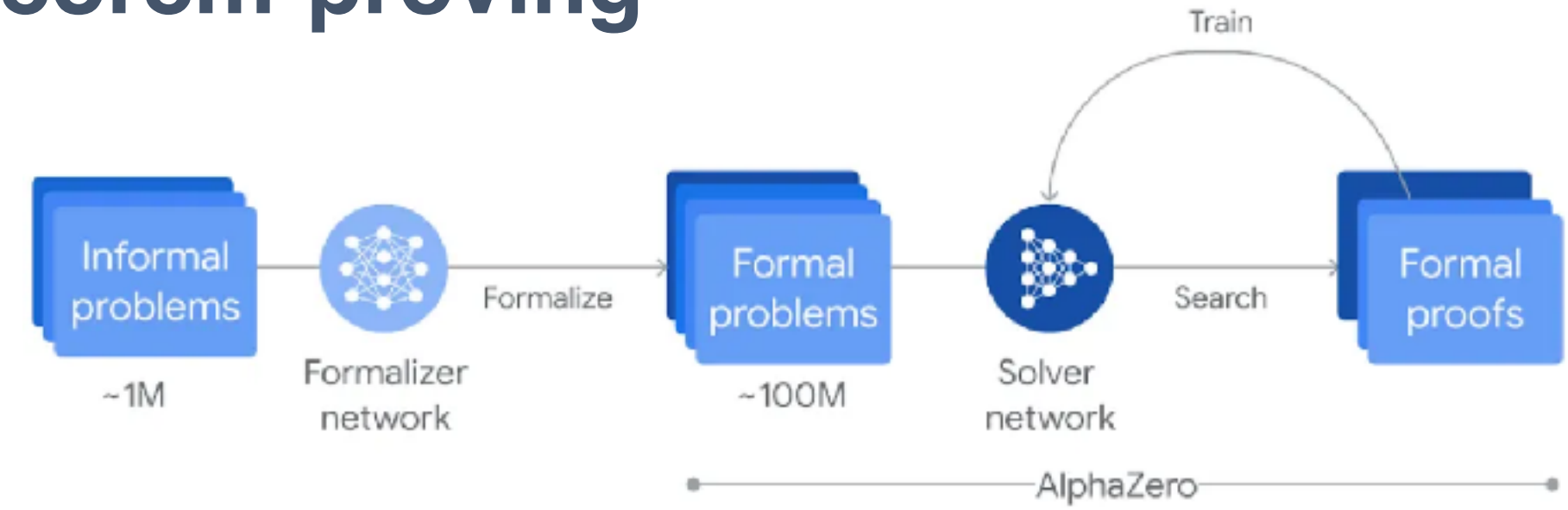
[Figure from *Formal Mathematical Reasoning: A New Frontier in AI*. Yang et al. 2

Neural Theorem Proving



[Figure from *Formal Mathematical Reasoning: A New Frontier in AI*. Yang et al. 2

Alphaproof: Reinforcement learning for theorem-proving



- Learn from both successes (proofs) and failures (disproofs).
- Misformalized problems are still helpful for learning
- Complement RL training with **test-time RL** on problem variants.

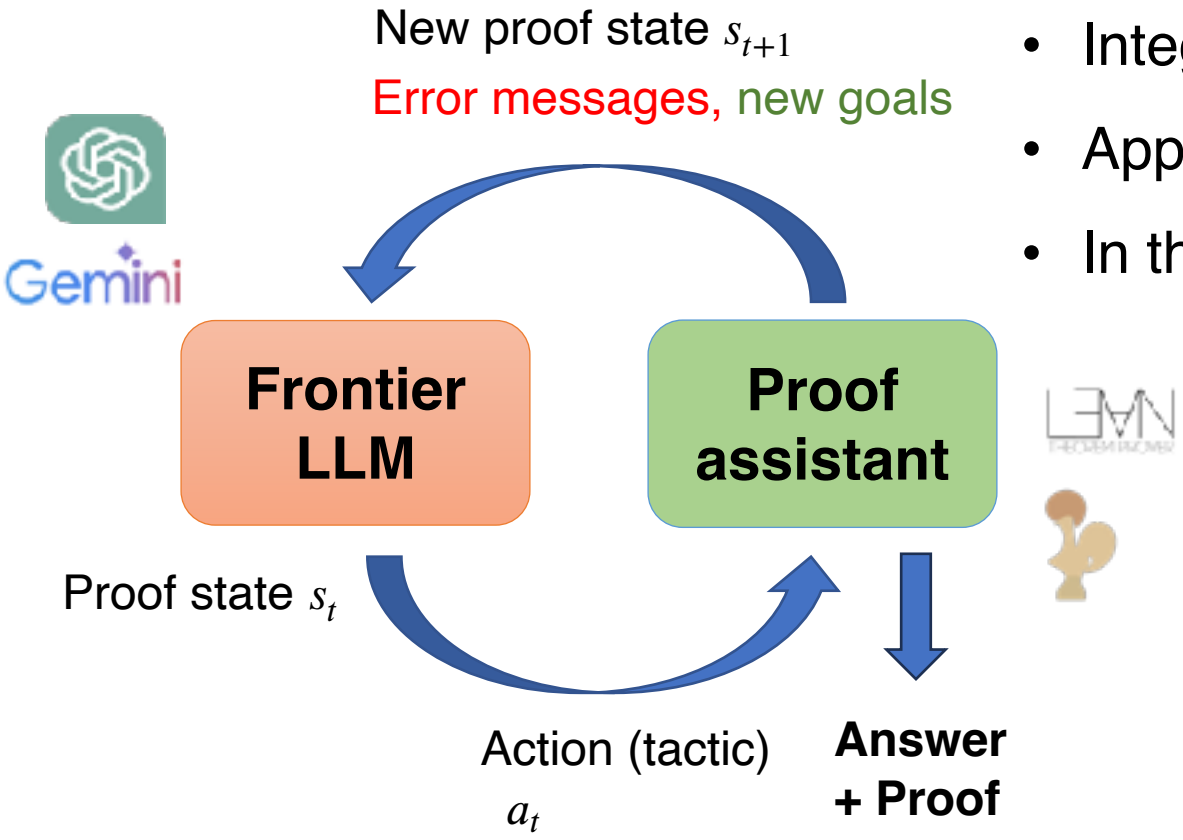
Google DeepMind "AI achieves silver-medal standard solving International Mathematical Olympiad problems", 2024.

An In-Context Learning Agent for Formal Theorem-Proving

Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, Swarat Chaudhuri.
Conference on Language Models, 2024.

Amitayush

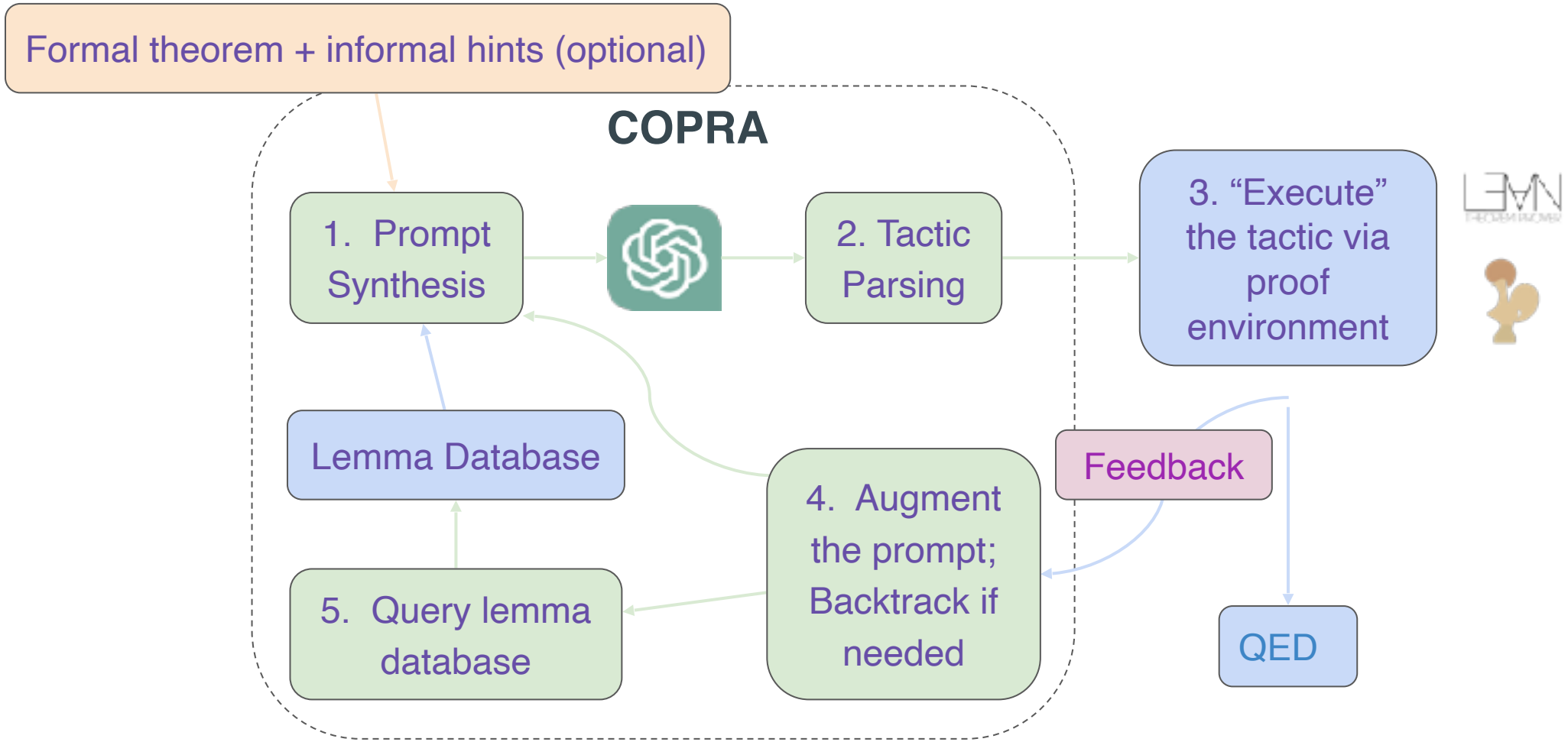
Copra: LLM agents for formal theorem-proving



- Immediately leverage advances in LLMs
- Integrate natural-language and formal reasoning
- Applicable even if there is no corpus of training problem
- In the longer run, source of traces for LLM finetuning.

Thakur, Tsoukalas, Wen, Xin, and Chaudhuri. **An In-Context Learning Agent for Formal Theorem-Proving.** COLM 2024. https://arxiv.org/pdf/2310.04353.pdf.

Copra: Theorem-proving via in-context learning



Prompting and action parsing

Serialized Proof State	[GOALS] [GOAL] 1 x * x % 2 = 0 [HYPOTHESES] 1 [HYPOTHESIS] x : ℕ [HYPOTHESIS] h : x % 2 = 0			[GOALS] [GOAL] 1 x % 2 * (x % 2) % 2 = 0 [HYPOTHESES] 1 [HYPOTHESIS] x : ℕ [HYPOTHESIS] h : x % 2 = 0
Stack & Failure Dictionary	[LAST STEP] intro h,	[STEPS] [STEP] intro h, [LAST STEP] rw h,	[STEPS] [STEP] intro h, [INCORRECT STEPS] [STEP] rw h, [LAST STEP] apply nat.mul_mod_right,	[STEPS] [STEP] intro h, [LAST STEP] rw nat.mul_mod,
Interaction Result	[SUCCESS] [END]	[ERROR MESSAGE] Got error in 'rw h,': error: rewrite tactic failed, did not find instance of the pattern in the target expression x % 2 [END]	[ERROR MESSAGE] Got error in 'apply nat.mul_mod_right,': error: invalid apply tactic, failed to unify x * x % 2 = 0 with ?m_1 * ?m_2 % ?m_1 = 0 [END]	[SUCCESS] [END]
LLM Response	Query #1	Query #2	Query #3	Query #4
	[RUN TACTIC] rw h, [END]	[RUN TACTIC] apply nat.mul_mod_right, [END]	[RUN TACTIC] rw nat.mul_mod, [END]	[RUN TACTIC] rw h, [END]

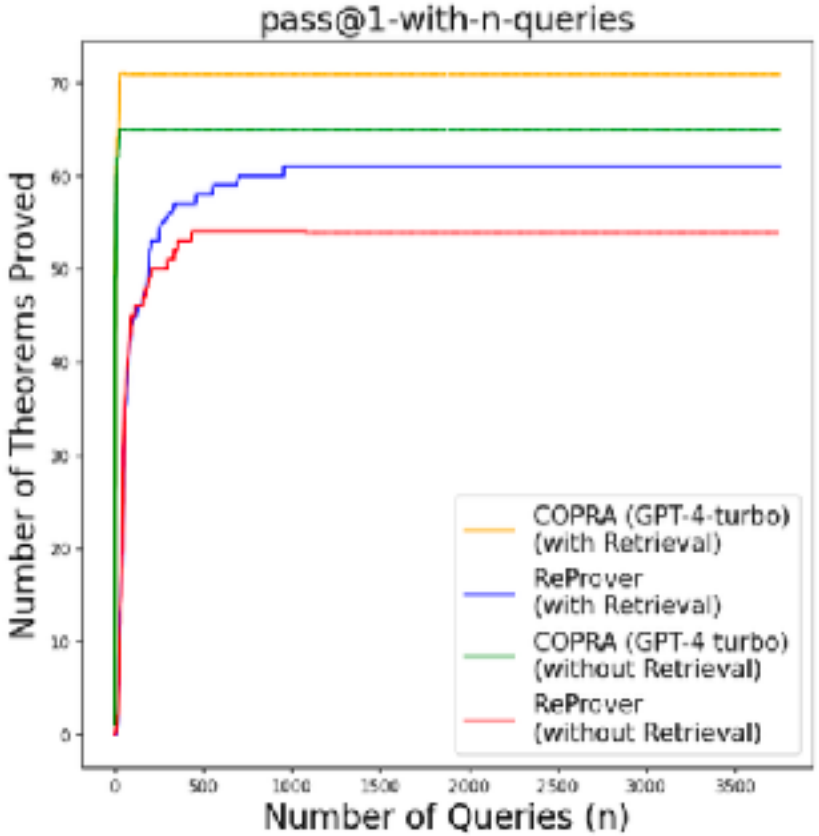
Copra vs. GPT-4

```
theorem mod_arith_2
(x : ℕ) : x % 2 = 0
     → (x * x) % 2 = 0
:=
begin
  intro h,
  rw nat.mul_mod,
  rw h,
  rw nat.zero_mul,
  refl,
end
```

A Lean theorem and a correct Copra-generated proof

```
begin
  intro h,
  have h1 : x = 2 * (x / 2)
     := (nat.mul_div_cancel' h).symm,
  rw h1,
  rw nat.mul_div_assoc _
    (show 2 ∣ 2, from dvd_refl _),
  rw [mul_assoc, nat.mul_mod_right],
end
```

A wrong “proof” generated by GPT-4



Evaluation on `miniF2F-test`		
Approach	***pass@k -with-n-queries*** $k \times n$ **(Timeout)**	**% proved**
Few-Shot (CodeLlama)	1 × 1 (600)	0.0%
Few-Shot (GPT 3.5)	1 × 1 (600)	2.8%
COPRA (CodeLlama)	1 × 500 (600)	5.73%
COPRA (GPT-3.5)	1 × 60 (600)	9.02%
Few-Shot	1 × 1 (600)	13.52%
Few-Shot ($T = 0.7$)	$60^{\ddagger}$ × 1 (600)	15.98%
REPROVER (- Retrieval)	1 × 1076 (600)	22.13%
COPRA (- Backtracking)	1 × 60 (600)	24.59%
REPROVER	1 × 3751 (600)	25.00%
LLEMMA-34b	1 × 3200 (600)	25.82%
LLEMMA-7b	1 × 3200 (600)	26.23%
COPRA	1 × 60 (600)	**26.63%**
COPRA (+ Retrieval)	1 × 60 (600)	**29.09%**
COPRA (+ Retrieval + Informal)	1 × 60 (600)	**29.92%**
COPRA (+ Retrieval + Informal)	1 × 100 (1200)	**30.74%**

Integrating natural-language and formal reasoning

IMO 1959, Problem 1:

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

Idea: Break the problem into subproblems and solve hierarchically.

1. Ask for an informal solution

Give the steps to solve the following problem:

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

O3-mini

To prove that the fraction

$$\frac{21n+4}{14n+3}$$

is irreducible for every natural number n, we need to show that the numerator and the denominator have no common factor other than 1. One effective method is to use the Euclidean algorithm. Here are the steps:

1. **Define the Greatest Common Divisor (gcd):**
   Let:
   $$d = \gcd(21n + 4, 14n + 3).$$
   Our goal is to show that $d = 1$.
2. **Subtract to simplify:**
   Compute the difference between the numerator and the denominator:
   $$(21n + 4) - (14n + 3) = 7n + 1.$$
   Since d divides both $21n + 4$ and $14n + 3$, it must also divide their difference:
   $$d \mid (7n + 1).$$
4. **Apply the Euclidean Algorithm:**
   Now, consider the gcd of $14n + 3$ and $7n + 1$. We can express:
   $$\gcd(21n + 4, 14n + 3) = \gcd(14n + 3, 7n + 1).$$
   Next, form a linear combination of $14n + 3$ and $7n + 1$. Notice that:
   $$(14n + 3) - 2(7n + 1) = 14n + 3 - 14n - 2 = 1.$$
4. **Conclude that the gcd is 1:**
   Since d divides both $14n + 3$ and $7n + 1$, it must also divide any linear combination of these numbers. In particular, d divides 1:
   $$d \mid 1.$$
   The only positive divisor of 1 is 1 itself, so we conclude that:
   $$d = 1.$$
5. **Final Conclusion:**
   Because the greatest common divisor of $21n + 4$ and $14n + 3$ is 1, the fraction is irreducible for every natural number n.

This completes the proof.

2. Split the theorem into relevant sub-goals

Now given this lemma in Lean 4:

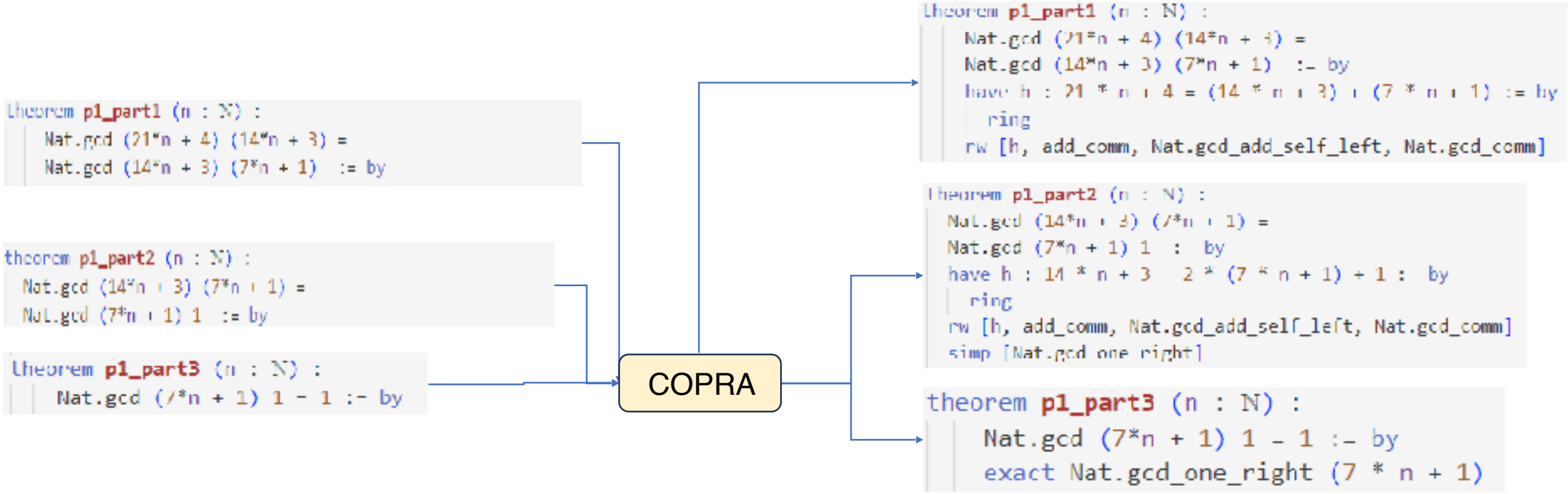
```
theorem imo_1959_p1
  (n : ℕ)
  (h₀ : 0 < n) :
  Nat.gcd (21*n + 4) (14*n + 3) = 1 := by
```

break it into sub-lemmas, don't try to write the proof, just write subgoals needed to prove this. Write a Lean 4 formal statement for the same

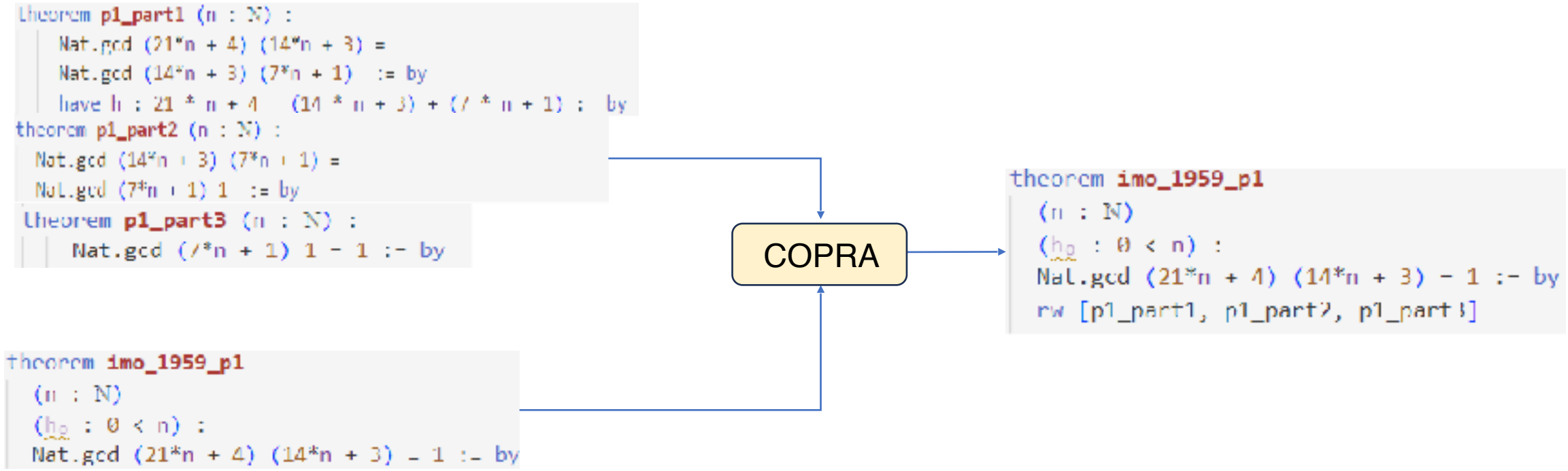
O3-mini

```
-- 1. Show that gcd (21*n + 4) (14*n + 3) = gcd (14*n + 3) (7*n + 1)
theorem gcd_step1 (n : ℕ) :
    Nat.gcd (21*n + 4) (14*n + 3) =
    Nat.gcd (14*n + 3) (7*n + 1) := by
     proof omitted
  sorry

  2. Show that gcd (14*n + 3) (7*n + 1) = gcd (7*n + 1) 1
theorem gcd_step2 (n : ℕ) :
    Nat.gcd (14*n + 3) (7*n + 1) =
    Nat.gcd (7*n + 1) 1 := by
  -- proof omitted
  sorry

-- 3. Finally, gcd (7*n + 1) 1 = 1
theorem gcd_step3 (n : ℕ) :
    Nat.gcd (7*n + 1) 1 = 1 := by
  -- proof omitted
  sorry

  Main theorem, combining the above
theorem imo_1959_p1 (n : ℕ) (h₀ : 0 < n) :
    Nat.gcd (21*n + 4) (14*n + 3) = 1 := by
```

3. Ask Copra to solve the subproblems one by one.



4. Add all relevant lemmas already proved in the system prompt, along with the informal proof.



o3-mini cannot solve the problem after splitting it into subproblems.

- `o3-mini` hallucinated a lemma "`Nat.gcd_sub_right`" while writing the proof.
- Copra can fix this hallucination lemma using Lean feedback.

Application in Formal Verification

Formal verification:

- Mathematically model a system as a set of *definitions*
- Model system properties as *theorems* involving these definitions
- Prove the theorems

Example: Compiler Verification

Language: Syntax + semantics

- **Syntax:** A grammar defining the form of programs
- **Semantics:** Rules defining executions of programs,

Compiler: A set of rules translating programs from one language to another.

Verification: Prove that the rules preserve semantics.

1. Define the source language

```
(*Define the source language first*)
Inductive binop : Set := Plus | Times.

Inductive exp : Set :=
 Const : nat -> exp
  (* Binop is a function which takes exp and exp and gives
exp.*)
| Binop : binop -> exp -> exp -> exp.

Definition binopDenote (b : binop) : nat -> nat -> nat :=
match b with
     | Plus => plus
     | Times => mult
end.

Fixpoint expDenote (e: exp) : nat :=
    match e with
     | Const n => n
     | Binop b e1 e2 =>
       (binopDenote b) (expDenote e1) (expDenote e2)
    end.
```

1. Define the source language

```
(*Define the source language first*)
Inductive binop : Set := Plus | Times.

Inductive exp : Set :=
 Const : nat -> exp
  (* Binop is a function which takes exp and exp and gives
exp.*)
| Binop : binop -> exp -> exp -> exp.

Definition binopDenote (b : binop) : nat -> nat -> nat :=
match b with
     | Plus => plus
     | Times => mult
end.

Fixpoint expDenote (e: exp) : nat :=
    match e with
     | Const n => n
     | Binop b e1 e2 =>
       (binopDenote b) (expDenote e1) (expDenote e2)
    end.
```

1. Define the source language

```
(*Some examples in Source Language*)
Eval simpl in expDenote (Const 42). (* 42 *)
Eval simpl in expDenote (Binop Plus (Const 2) (Const 2)). (* 4 *)
Eval simpl in expDenote (Binop Times (Binop Plus (Const 2) (Const 2)) (Const 7)). (*
28 *)
```

2. Define the target language

```
(*Define the target language*)

(*Instructions can be either constants or binary operation*)
Inductive instr: Set :=
| iConst : nat -> instr
| iBinop : binop -> instr.

Definition prog := list instr. (*Program is a list of instructions*)

(*Instruction either pushes a constant to the stack or applies binop
on two elements on the stack*)
Definition stack := list nat.

Definition instrDenote (i : instr) (s: stack): option stack :=
match i with
  | iConst n => Some (n :: s)
  | iBinop b =>
    match s with
      | arg1 :: arg2 :: s' => Some ((binopDenote b) arg1 arg2 :: s')
      | _ => None
    end
end.
```

2. Define the target language

```
Fixpoint progDenote (p : prog) (s: stack) : option stack :=
match p with (*Run instructions one by one*)
  | nil => Some s
  | i::p' =>
    match instrDenote i s with
      | None => None
      | Some s' => progDenote p' s'
    end
end.

(*Check the execution of program once*)
Eval simpl in progDenote (iConst 32:: iConst 42 :: iBinop Plus :: nil)
nil.
(* Some (74 :: nil) *)
```

3. Define the compiler

```
(*Translation from source to target language i.e. Compiler*)
Fixpoint compile (e : exp): prog :=
    match e with
        | Const n => iConst n::nil
        | Binop b e1 e2 => compile e2 ++ compile e1 ++ iBinop b :: nil
    end.
```

```
(*Convert the source language to target assembly language *)
Eval simpl in compile (Binop Plus (Const 2) (Const 3)).
(* iConst 3 :: iConst 2 :: iBinop Plus :: nil *)
```

4. Define a correctness theorem

```
(*Ensure that the compiled program compiles correctly for all possible expr *)

Theorem compile_correct:
     forall e, progDenote (compile e) nil = Some (expDenote e::nil).
```

5. Ask Copra to produce the proof

```
(*Ensure that the compiled program compiles correctly for all possible expr *)

Theorem compile_correct:
     forall e, progDenote (compile e) nil = Some (expDenote e::nil).
```

Failure! ❌

5(a). Ask the LLM to invent a lemma

We want to prove the following theorem:
Theorem compile_correct:
forall e, progDenote (compile e) nil = Some (expDenote e::nil).

However, this is a difficult theorem to prove, and we need to strengthen the inductive bias for it to work. Can you generate a theorem/lemma which we can use to prove compile_correct?

5(a). Ask the LLM to invent a lemma

Thought for 7 seconds

Yes, to prove `compile_correct`, we can strengthen the inductive hypothesis by generalizing the theorem. Specifically, we can prove the following lemma:

coq

Copy code

```
Lemma compile_correct_aux: forall e s,
    progDenote (compile e) s = Some (expDenote e :: s).
```

This lemma states that for any expression `e` and any stack `s`, running the compiled program of `e` on stack `s` results in the value of `e` pushed onto `s`. Once we have this lemma, we can easily prove `compile_correct` by setting `s` to `nil`.

Alternatively, we can generalize even further by considering an additional program `p` that runs after the compiled code:

coq

Copy code

```
Lemma compile_correct_gen: forall e p s,
    progDenote (compile e ++ p) s = progDenote p (expDenote e :: s).
```

5(b). Sample a proposed lemma

Target theorem:

```
(*Ensure that the compiled program compiles correctly for all possible expr *)
Theorem compile_correct: forall e, progDenote (compile e) nil = Some (expDenote e::nil).
```

Auxiliary lemma:

```
(*Prove if we can compile one instruction correctly first*)
Lemma compile_one_instr: forall e p s, progDenote (compile e ++ p)
s = progDenote p (expDenote e :: s).
```

5(c). Auto-generate proof of `compile_one_instr`

```
Lemma compile_one_instr : forall e p s, progDenote (compile e ++
p) s = progDenote p (expDenote e :: s).
Proof.
    induction e using exp_ind.
    intros p s.
    unfold progDenote, compile, expDenote.
    simpl.
    reflexivity.
    intros p s.
    simpl.
    rewrite app_assoc_reverse.
    rewrite <- app_assoc.
    rewrite IHe2.
    rewrite IHe1.
    reflexivity.
Qed.
```

5(d). Now auto-generate the proof of `compile_correct`

```
Theorem compile_correct : forall e, progDenote (compile e) nil =
Some (expDenote e::nil).
Proof.
    destruct e.
    - simpl. reflexivity.
    - simpl. rewrite compile_one_instr. rewrite compile_one_instr.
reflexivity.
Qed.
```

Summary: Mathematical Discovery with LLM Agents

Language is an extraordinary powerful tool for mathematical reasoning.

Frontier LLMs can prove nontrivial formal theorems with proof assistant feedback.

Future work should expand these ideas in conjecturing and modeling tasks.

Question: Can prover-LLM interaction be entirely pushed to training time?

- The prover teaches the LLM student about compositionality, type-safety,…
- Eventually, the teacher “retires”.

Open Challenges and Future Directions

Formal Mathematical Reasoning: A New Frontier in AI

Kaiyu Yang[1], Gabriel Poesia[2], Jingxuan He[3], Wenda Li[4], Kristin Lauter[1], Swarat Chaudhuri[5], Dawn Song[3]
[1]Meta FAIR, [2]Stanford University, [3]UC Berkeley, [4]University of Edinburgh, [5]UT Austin

https://arxiv.org/abs/2412.16075

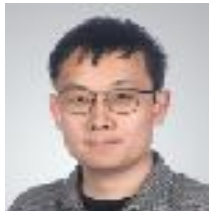

Kaiyu Yang


Dawn Song


Gabriel Poesia


Wenda Li

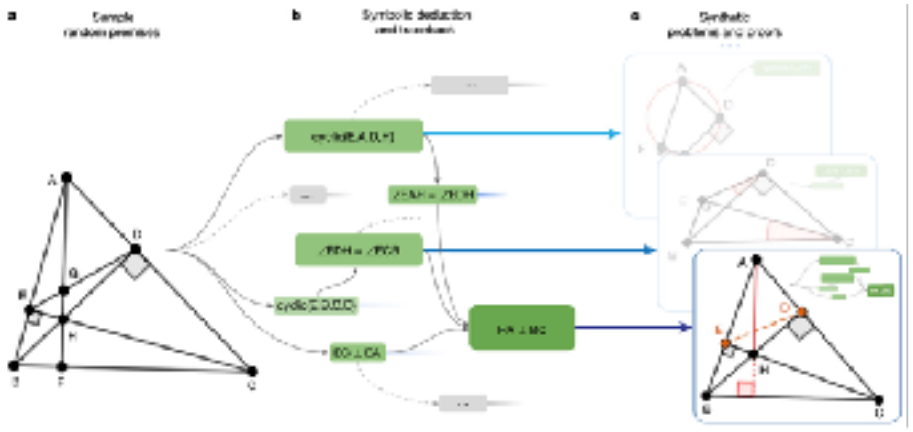

Kristin Lauter


Jingxuan He

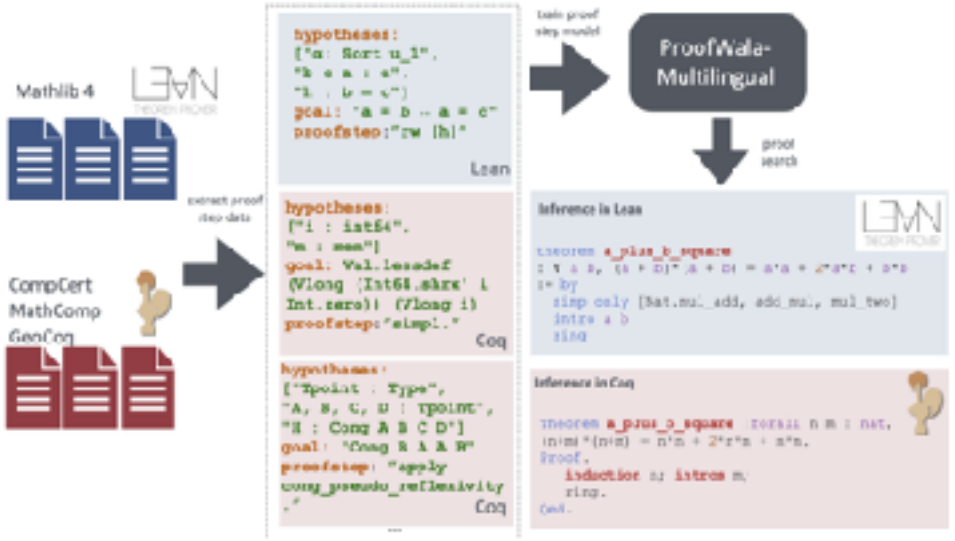
Open Challenges and Future Directions

Overcoming the shortage of high-quality proof data

- **Strategies:** Synthetic data, multilingual data, crowdsourcing



[Trinh et al., Alphageometry]

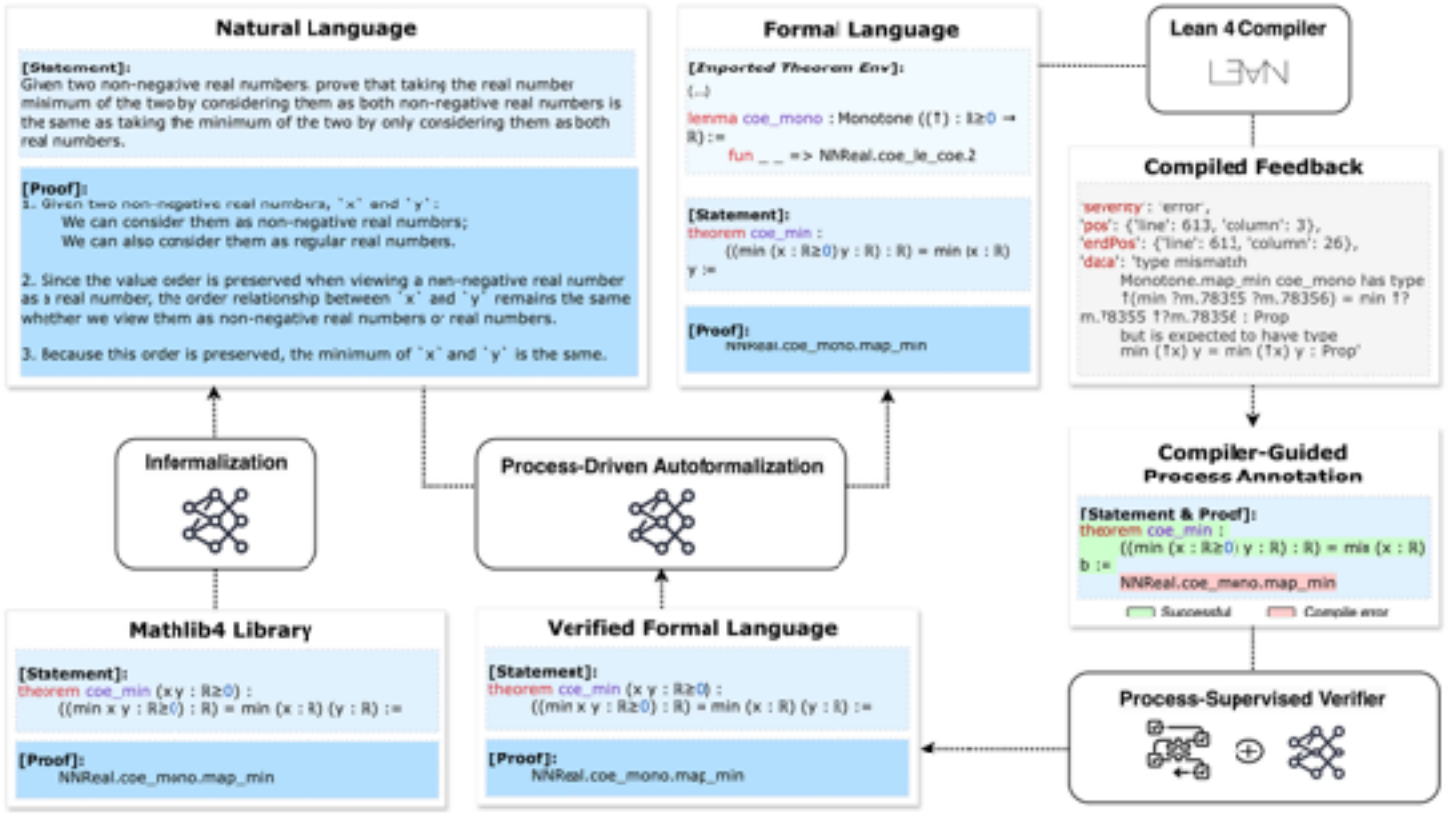


[Thakur et al., Proofwala]

Open Challenges and Future Directions

Autoformalization, which is hard due to insufficient high-quality paired data

- **Strategies:** Formalization with proof assistant feedback

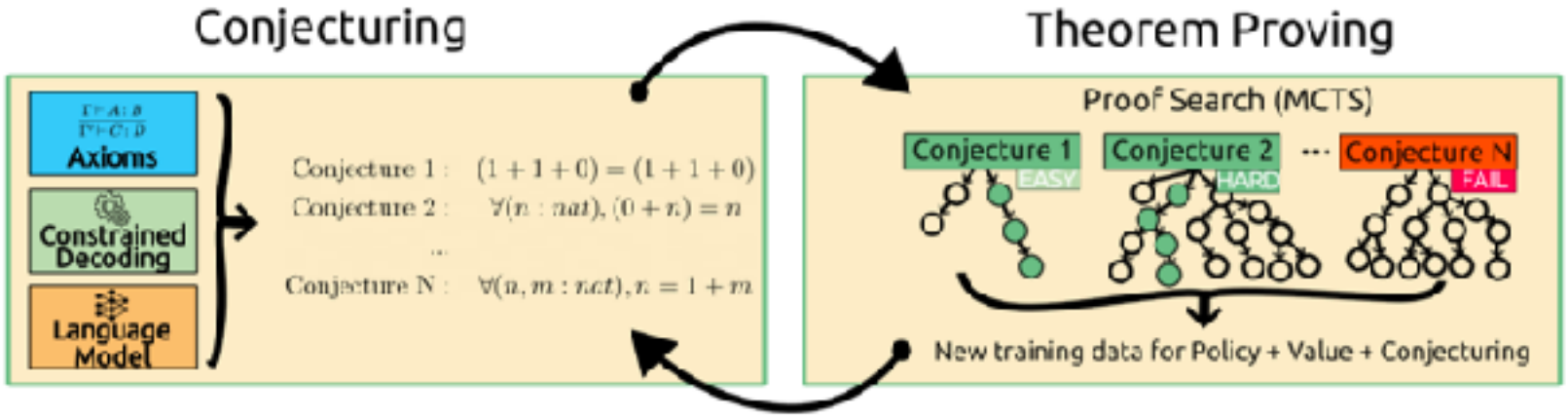


[Lu et al., Process-Driven Autoformalization]

Open Challenges and Future Directions

Conjeturing and open-ended exploration

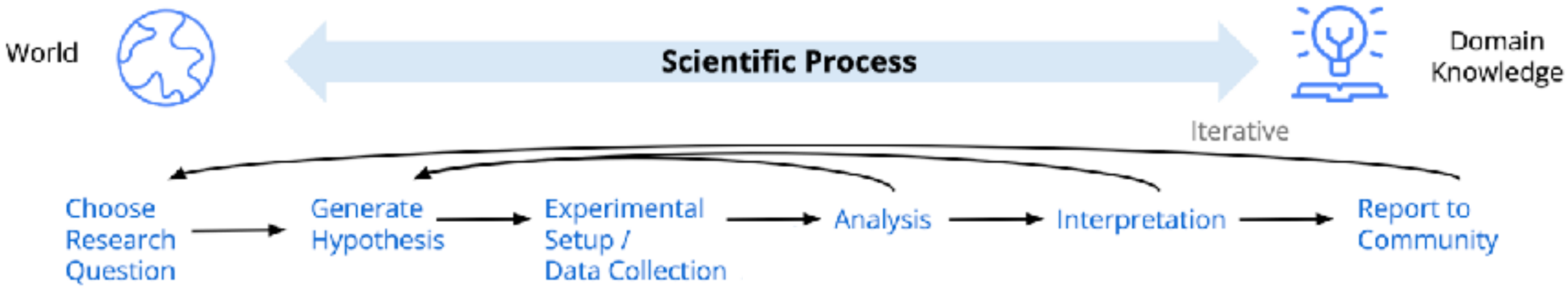
Strategy: Self-play between a conjecturer and a prover



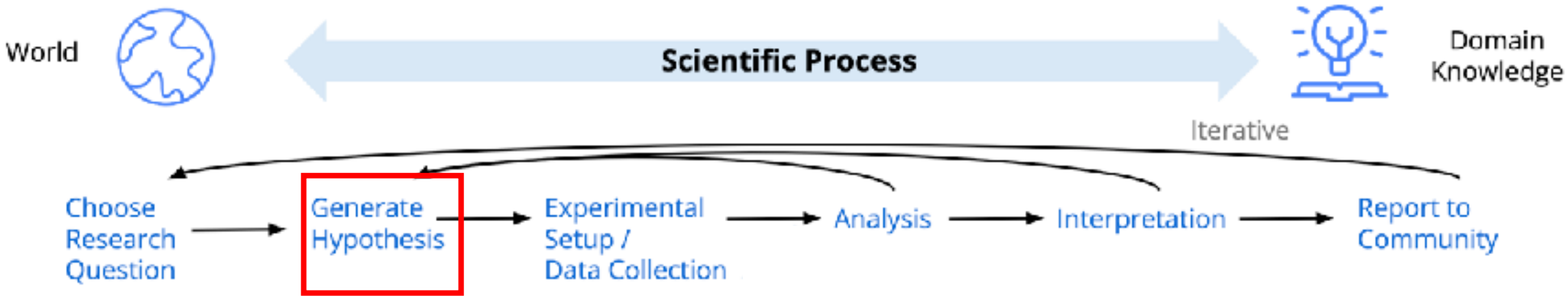
[Poesia et al., Minimo]

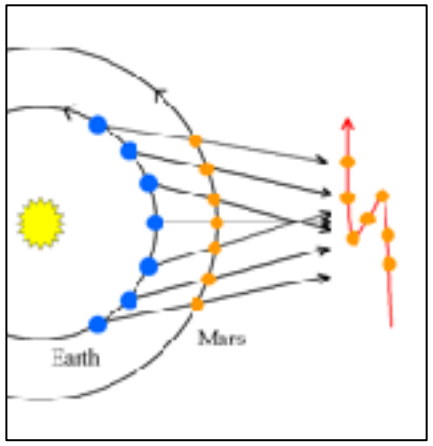
AI for Scientific Discovery

Lifecycle of a scientific process



Lifecycle of a scientific process



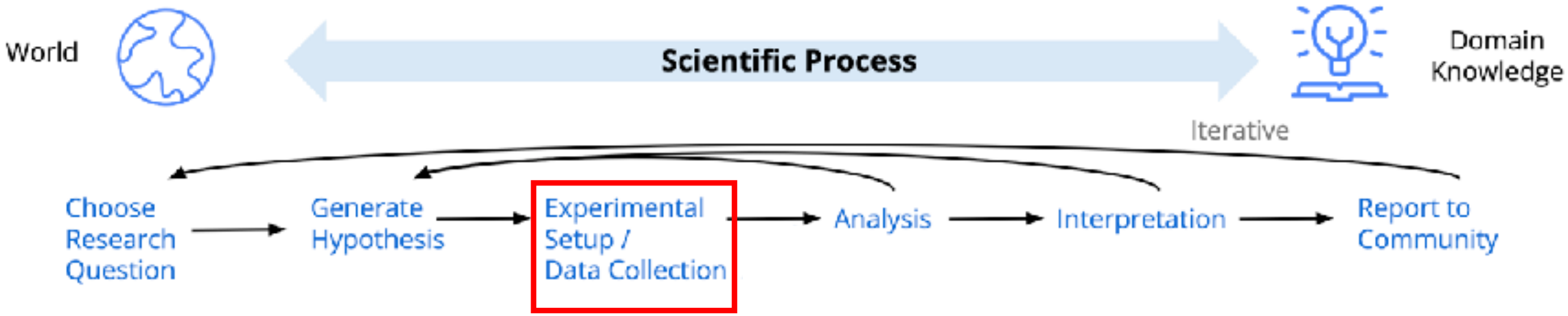


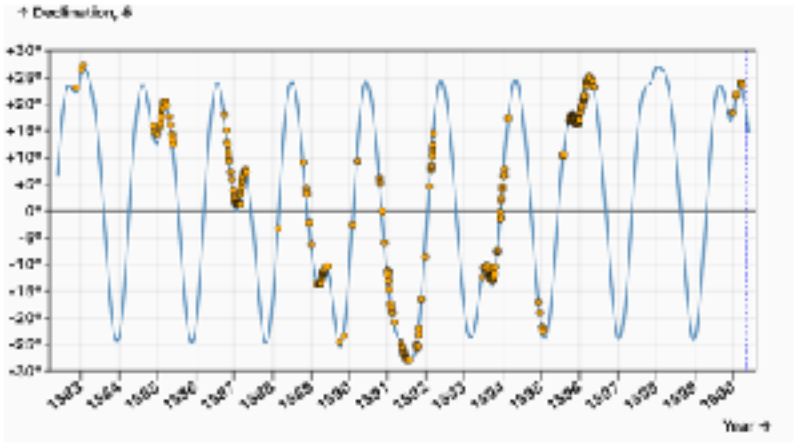
Problem: Apparent Retrograde Planetary Motion

Theory: Heliocentric Model

c. The Astronomical Revolution: Copernicus- Kepler-Borelli

Lifecycle of a scientific process

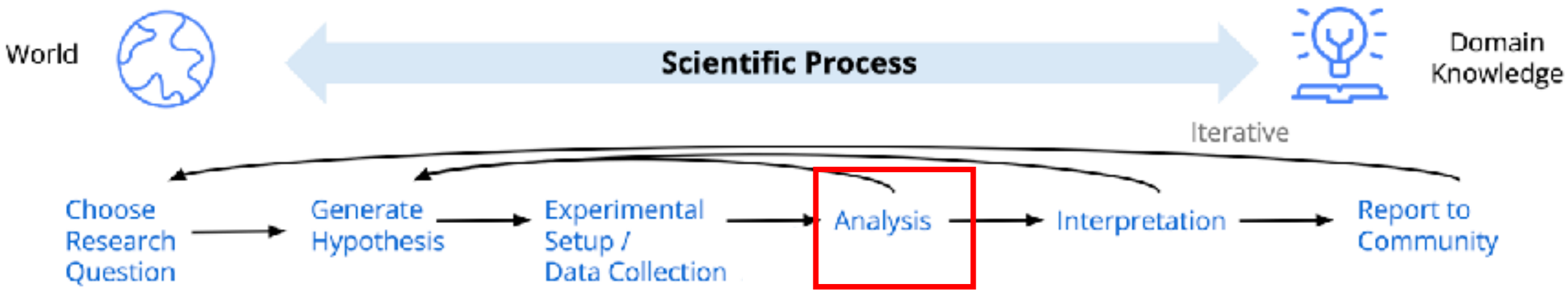




Tycho Brahe's observations of the planet Mars (1582-1600)

Data Collection: Sample data in regime of interest.

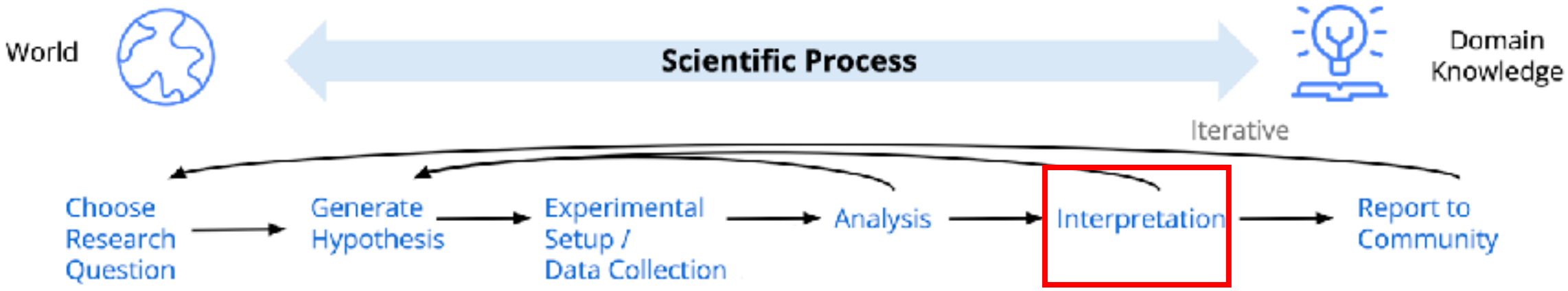
Lifecycle of a scientific process



Analysis: Kepler's Third Law

$$T^2 \propto r^3$$

Lifecycle of a scientific process



Analysis: Kepler's Third Law

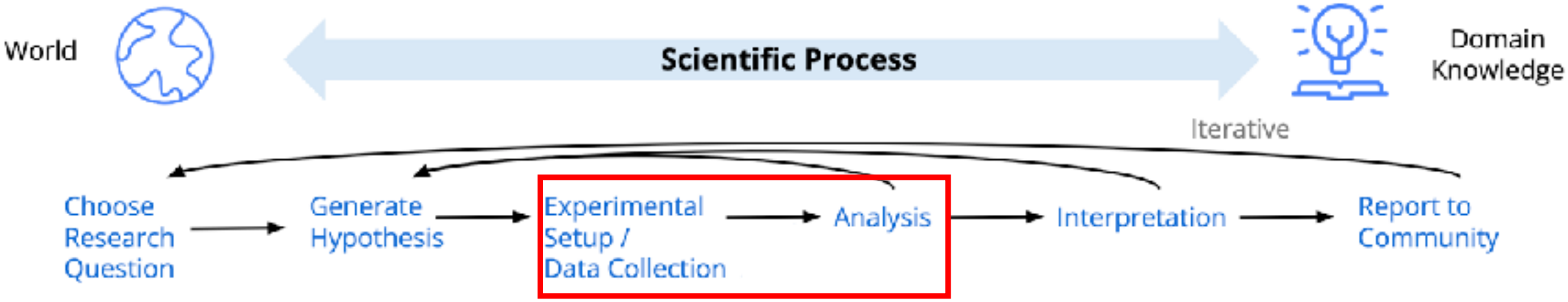
$$T^2 \propto r^3$$

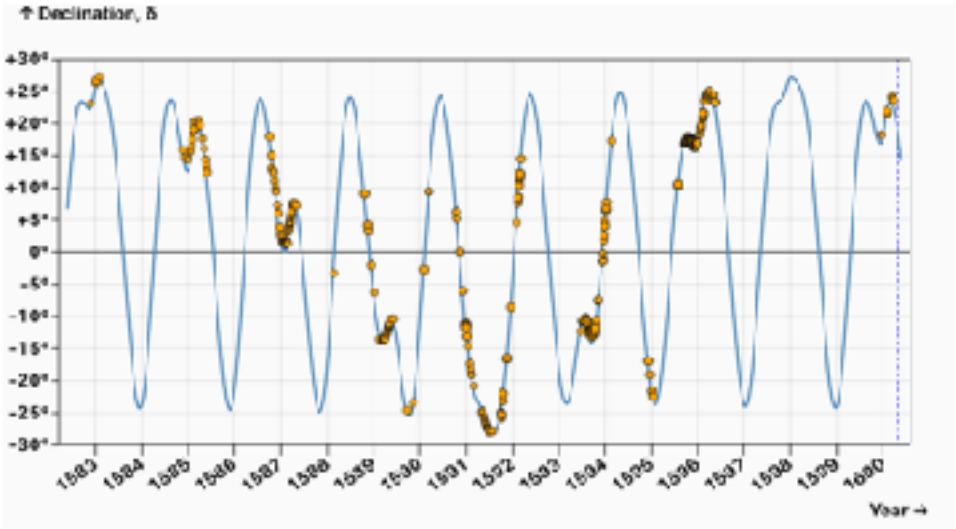
Interpretation: Newton's Law of Gravitation

$$mr\left(\frac{2\pi}{T}\right)^2 = G\frac{mM}{r^2}$$

c. The Astronomical Revolution: Copernicus- Kepler-Borelli

Symbolic regression





Kepler's Third Law

$$T^2 \propto r^3$$

c. The Astronomical Revolution: Copernicus- Kepler-Borelli

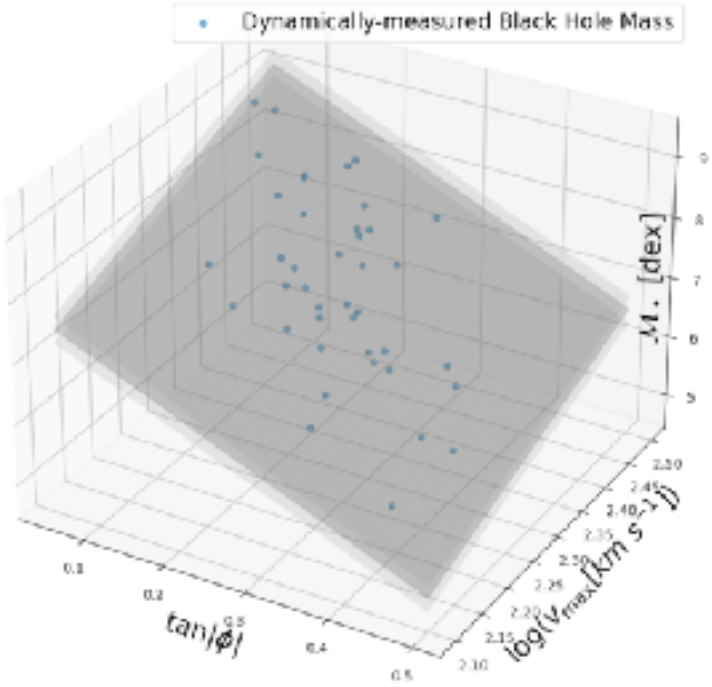
Symbolic regression algorithms

c. PySR. Miles Cranmer

Symbolic Regression with PySR

Credit: Miles Cranmer

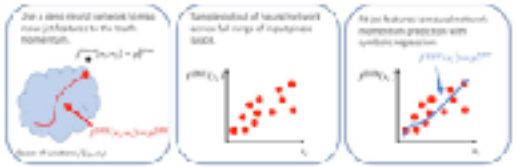
Symbolic Regression's impact



Discovery of a Planar Black Hole Mass Scaling Relation for Spiral Galaxies

Benjamin L. Davis [1], Zehao Jin [1]

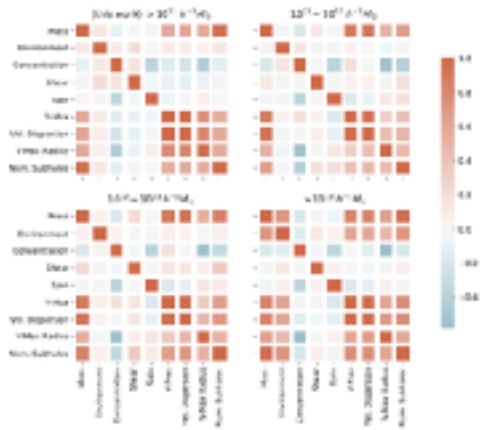
[1] Center for Astrophysics and Space Science, New York University Abu Dhabi



Interpretable machine learning methods applied to jet background subtraction in heavy-ion collisions

Tanner Mengel [1], Patrick Steffanic [1], Charles Hughes [1,2], Antonio Carlos Oliveira da Silva [1,2], Christine Nattrass [1]

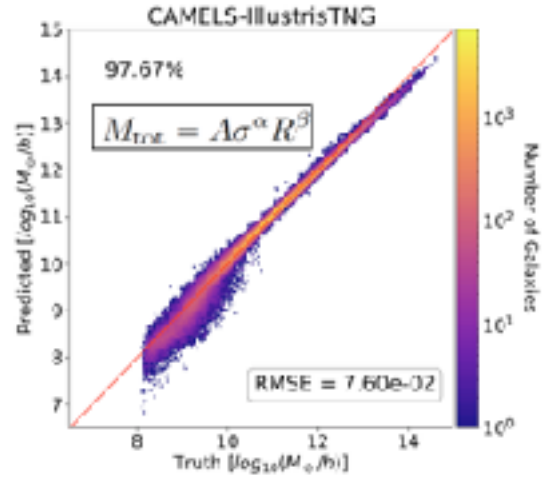
[1] University of Tennessee, Knoxville, [2] Iowa State University of Science and Technology



Modeling the galaxy-halo connection with machine learning

Ana Maria Delgado [1], Digvijay Wadekar [2,3], Boryana Hadzhiyska [1], Sownak Bose [1,4], Lars Hernquist [1], Shirley Ho [2,4,5,6]

[1] Center for Astrophysics | Harvard & Smithsonian, [2] New York University, [3] Institute for Advanced Study, [4] Flatiron Institute, [5] Princeton University, [6] Carnegie Mellon University, [7] Durham University



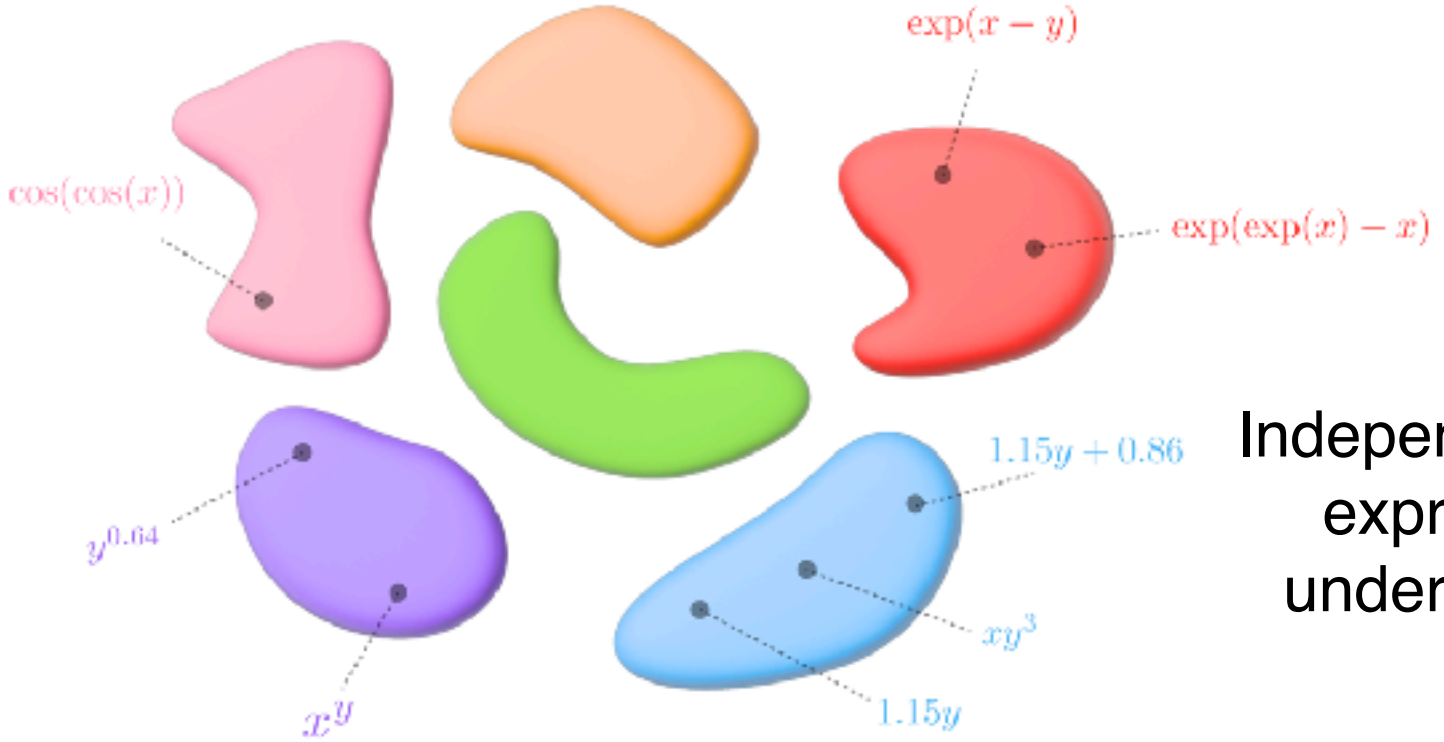
Finding universal relations in subhalo properties with artificial intelligence

Helen Shao [1], Francisco Villaescusa-Navarro [1,2], Shy Genel [2,3], David N. Spergel [2,1], Daniel Anglés-Alcázar [4,2], Lars Hernquist [3], Romeel Davé [5,7,8], Desika Narayanan [9,10], Gabriella Contardo [2], Mark Vogelsberger [11]

[1] Princeton University, [2] Flatiron Institute, [3] Columbia University, [4] University of Connecticut, [5] Center for Astrophysics | Harvard & Smithsonian, [6] University of Lombardi, [7] University of the Western Cape, [8] South African Astronomical Observatories, [9] University of Florida, [10] University of Florida Informatics Institute, [11] MIT

Credit: Miles Cranmer

Sketch of PySR's Exploration Space



Independent "islands" of expressions, each undergoing evolution

Insight: LLMs can increase exploration in relevant parts of the search space.

LaSR: Symbolic Regression with a Learned Concept Library

Arya Grayeli*, Atharva Sehgal*, Omar Costilla-Reyes, Miles Cranmer, Swarat Chaudhuri.
Neural Information Processing Systems, 2024.

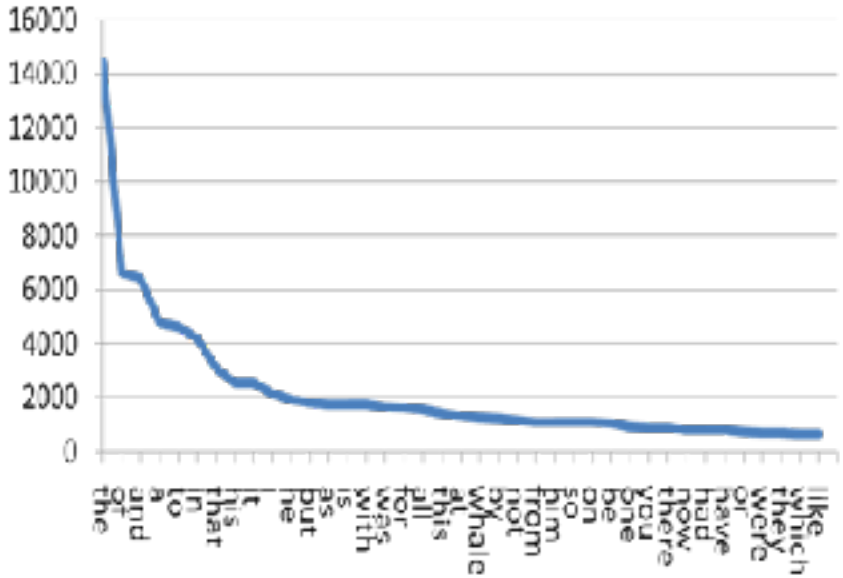
Arya

Atharva

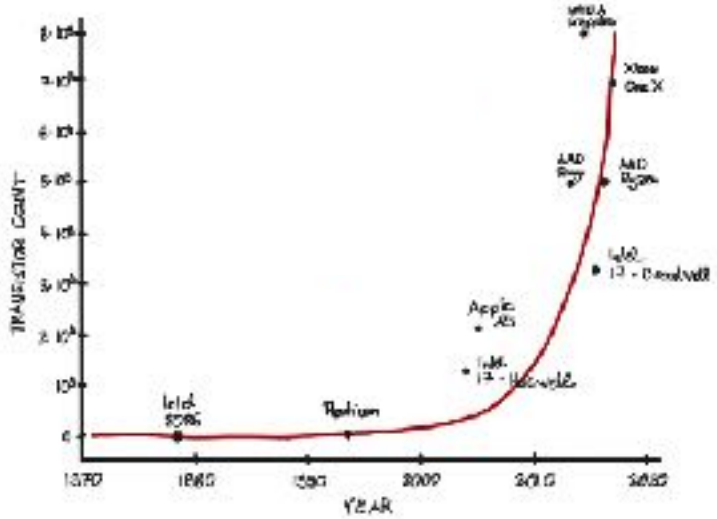
https://trishullab.github.io/lasr-web

What is a Concept?
Desiderata 1: Symbolic Abstraction

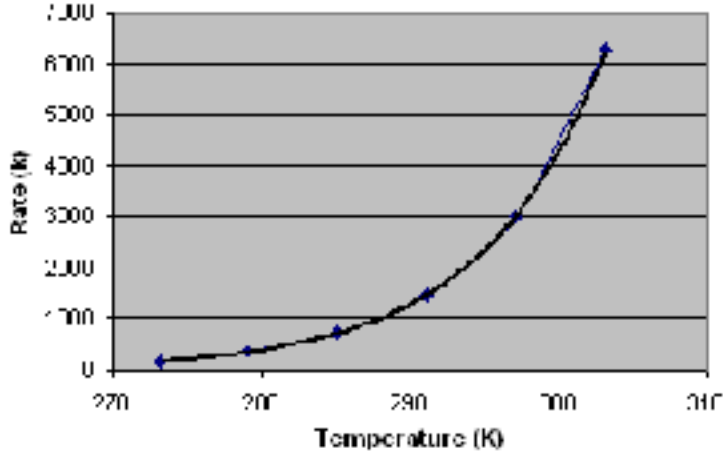
Zipf's Law



Moore's Law

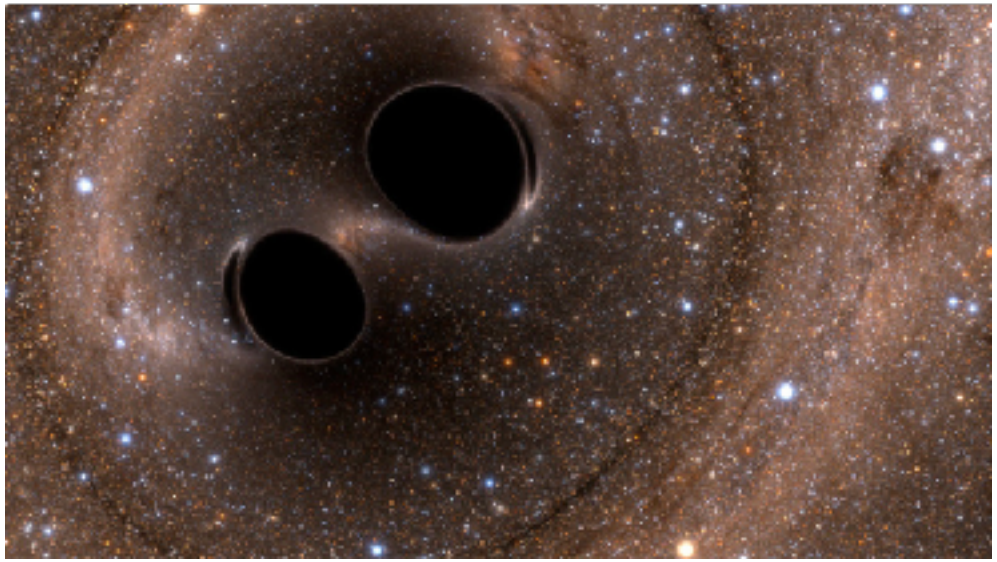


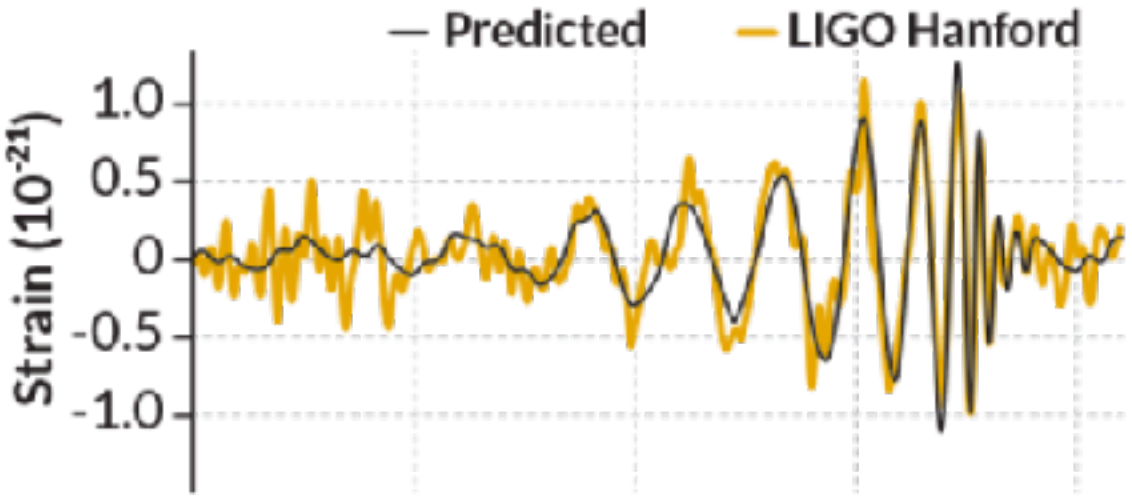
Arrhenius' Equation



$$y = ax^k + \epsilon \Leftrightarrow \textit{“Power Law Trend”}$$

What is a Concept?
Desiderata II : Symbolic Guidance





Concepts (by Physicist or LLM)

"Wave strain diminishes as distance increases"
"Wave strain has extraordinarily small magnitude"

Guide the search for

$$h = \frac{2G}{c^4}\frac{1}{r}\frac{\partial^2 Q}{\partial t^2}$$

h: strain
r: distance between the poles
Q: dipole moment

Joint concept and program learning

Given

A universe C of ***concepts***

- *"Wave strain diminishes as distance increases" , "power laws", "sinusoidal functions"*

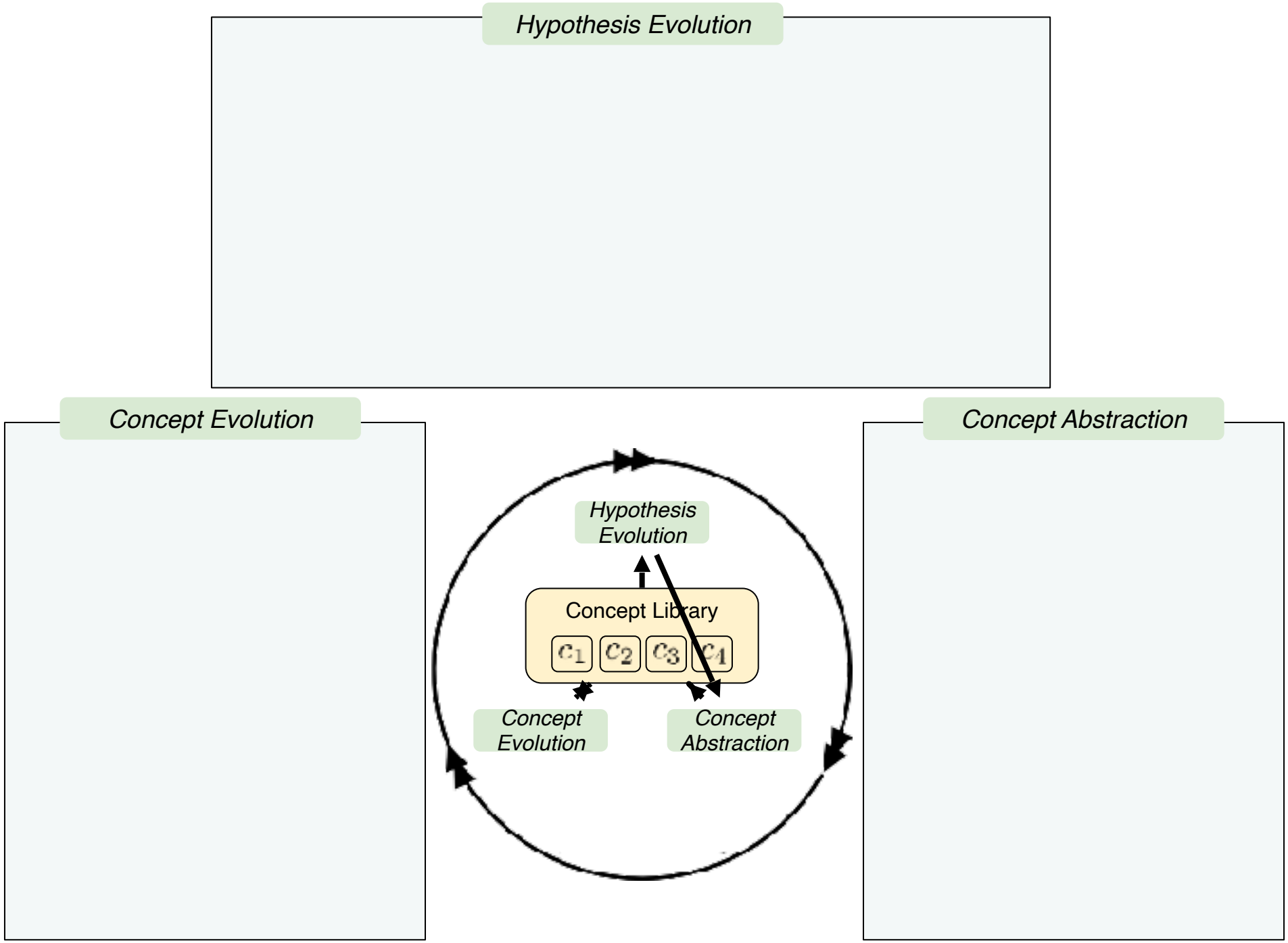
A dataset D

A space of programmatic hypotheses π

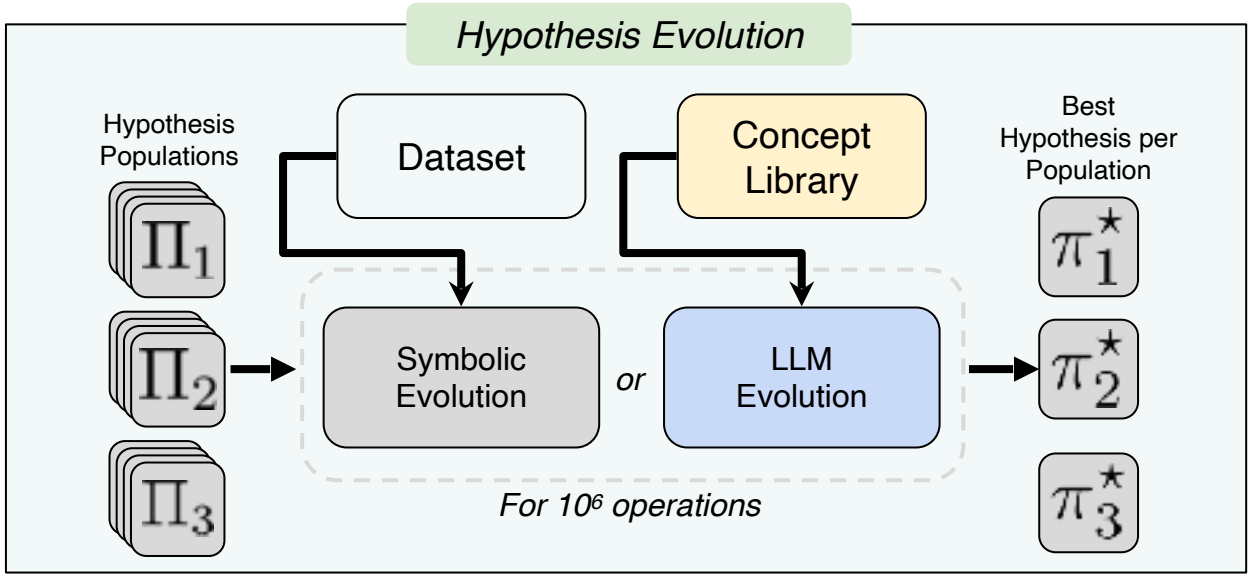
Solve

$$\arg\max_{\pi,\mathcal{C}} p(\pi,\mathcal{C}|\mathcal{D}) = \arg\max_{\pi,\mathcal{C}} \underbrace{p(\mathcal{D}|\pi)}_{\text{By execution}} \cdot \underbrace{p(\pi|\mathcal{C})}_{\text{By LLM}} \cdot \underbrace{p(\mathcal{C})}_{\text{By LLM}}$$

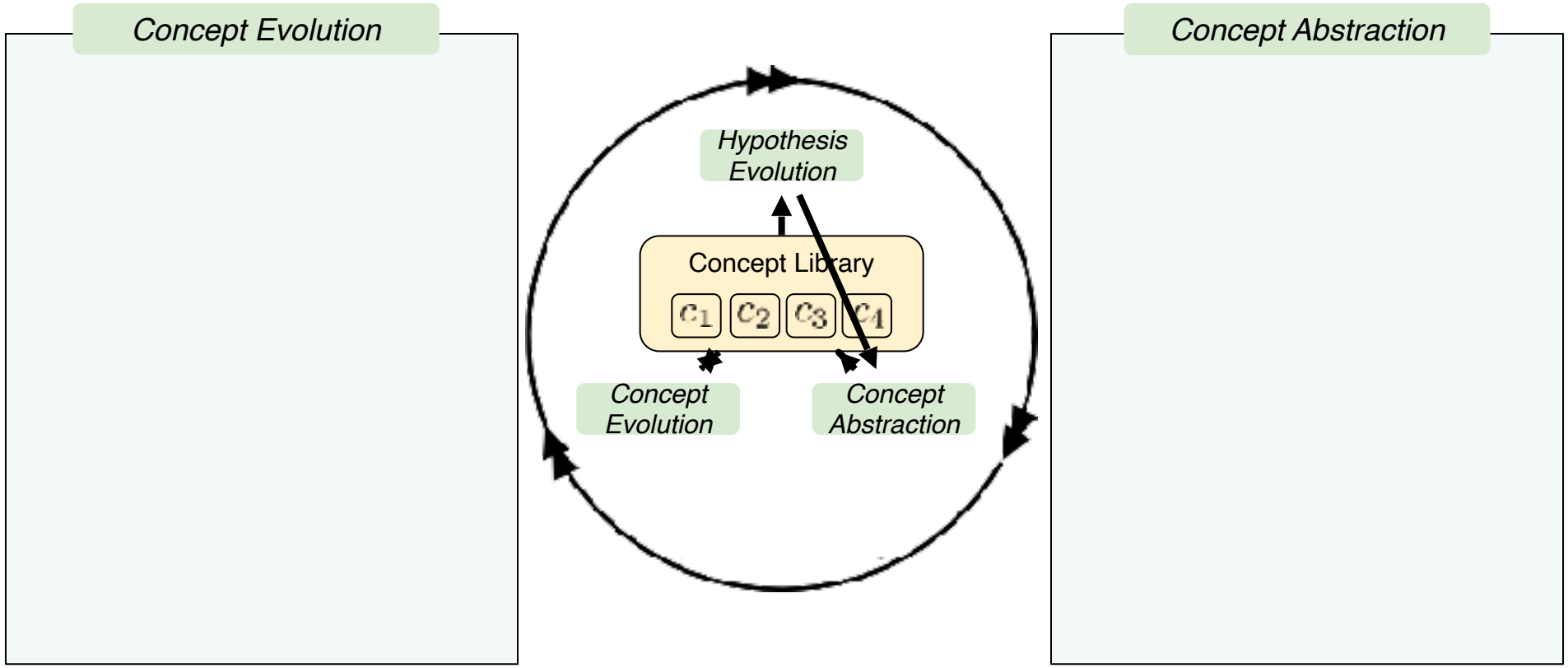
LaSR


Hypothesis Evolution
Concept Evolution
Concept Abstraction
Hypothesis Evolution
Concept Library
c_1 c_2 c_3 c_4
Concept Evolution
Concept Abstraction

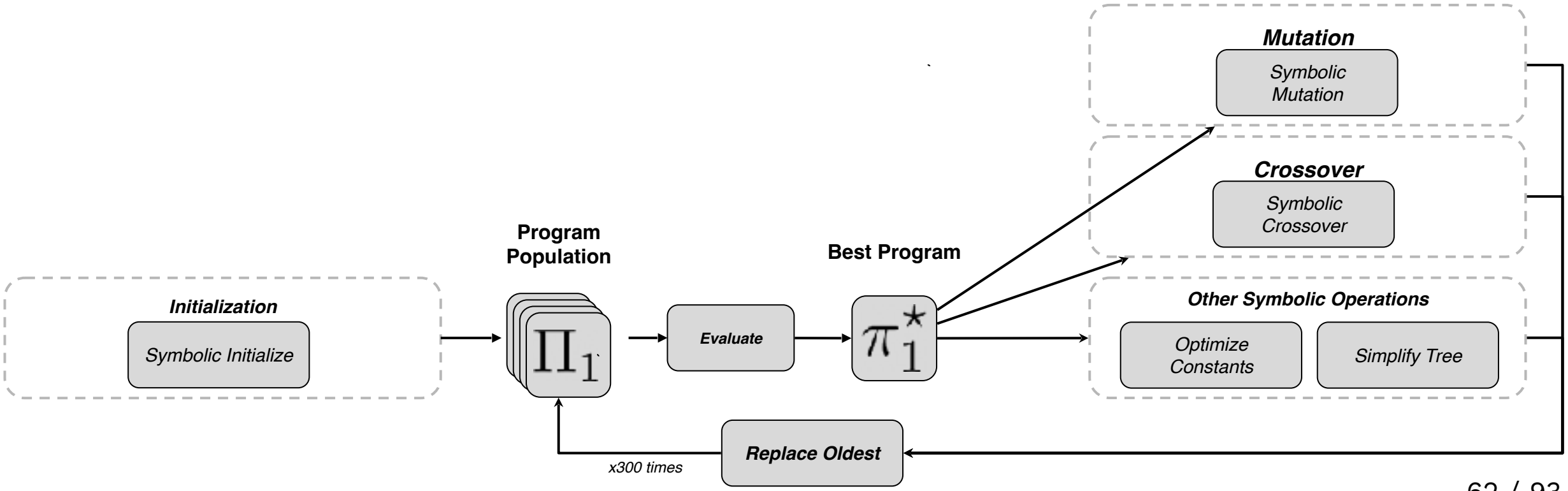
LaSR



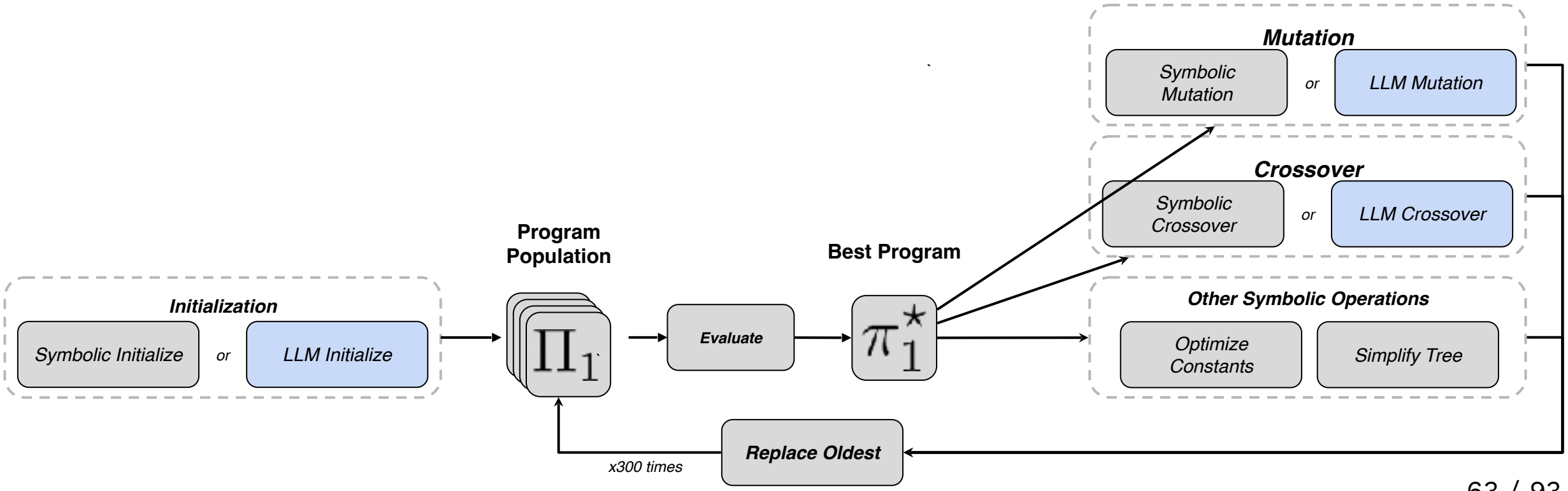
LLM Evolution provides *neural guidance* (over a language prior)



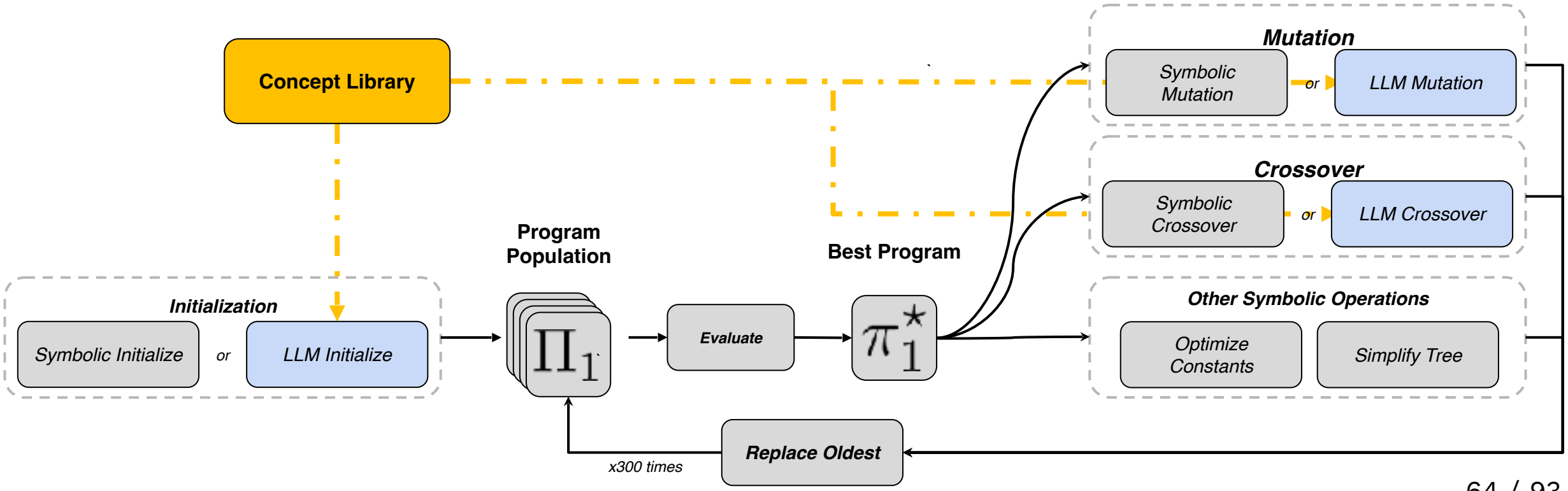
Hypothesis Evolution



Hypothesis Evolution



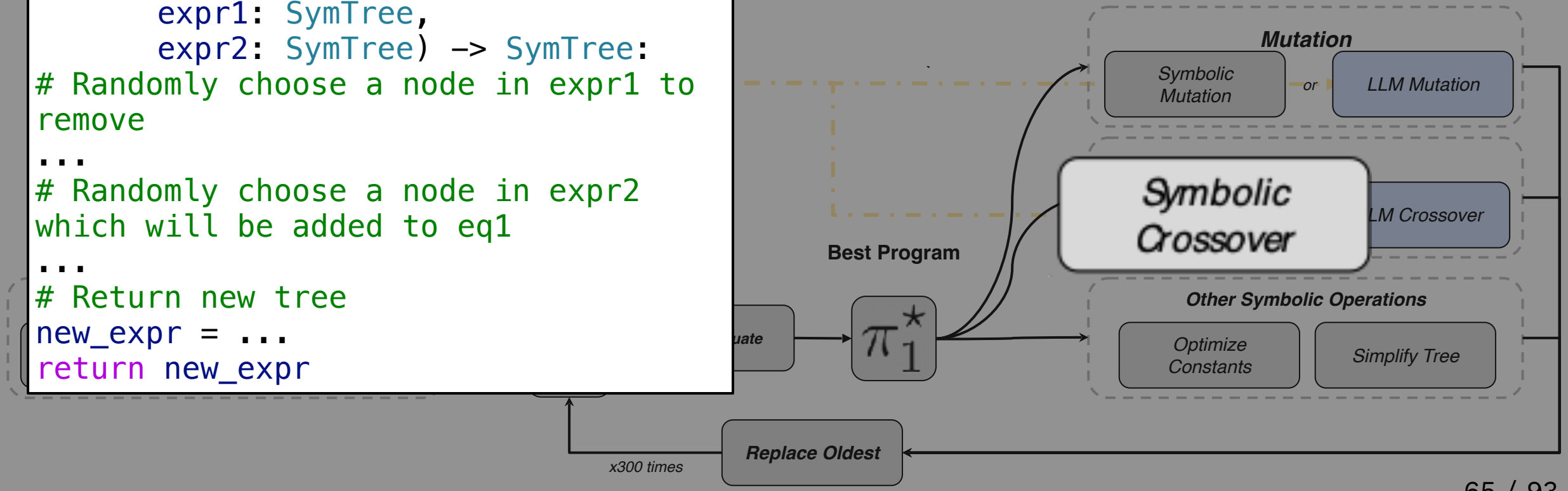
Hypothesis Evolution



Hypothesis Evolution

```python
import random

def crossover(
        expr1: SymTree,
        expr2: SymTree) -> SymTree:
# Randomly choose a node in expr1 to
remove
...
# Randomly choose a node in expr2
which will be added to eq1
...
# Return new tree
new_expr = ...
return new_expr
```



Hypothesis Evolution

(System) Header

You are a helpful assistant that recombines two mathematical expressions by following a few provided suggestions. You will be given three suggestions and two expressions to recombine.

An expression must consist of the following variables: `{{variables}}`. All constants will be represented with the symbol C. Each expression will only use these operators: `{{operators}}`.

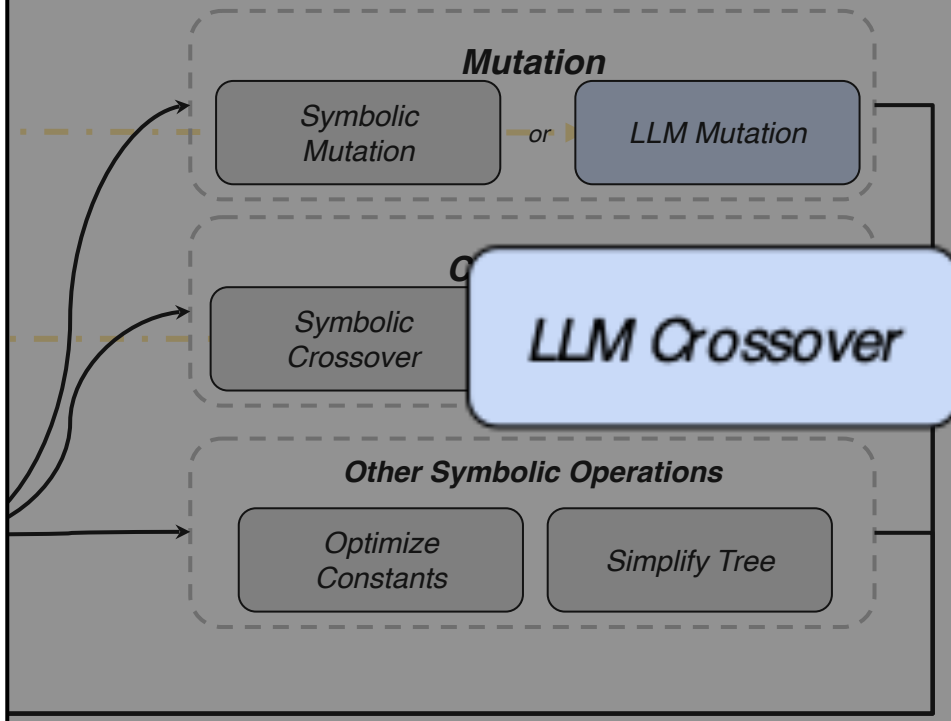
(User) Crossover Prompt

Suggestion 1: `{{assump1}}`
Suggestion 2: `{{assump2}}`
Suggestion 3: `{{assump3}}`
Expression 1: `{{expr1}}`
Expression 2: `{{expr2}}`

Propose `{{N}}` expressions that would be appropriate given the suggestions and expressions. Provide short commentary for each of your decisions. End with a JSON list that enumerates the proposed expressions following this format:

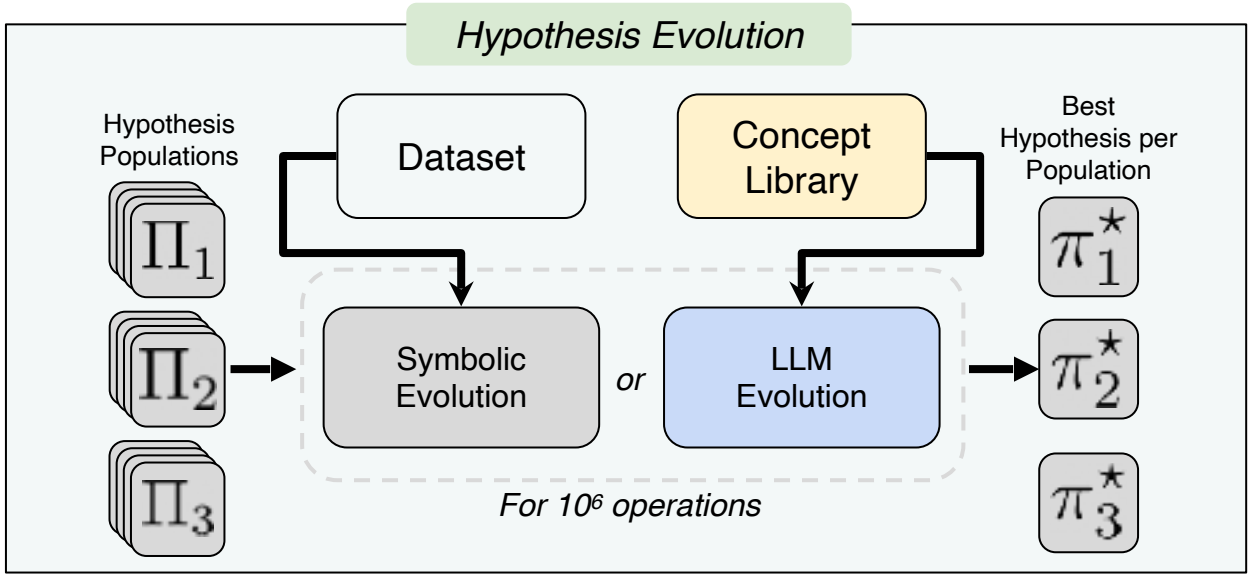
(User) JSON Formatting Instructions

```
```json
["expr1",
 "expr2",
 ...
 "expr{{N}}"
]
```
```

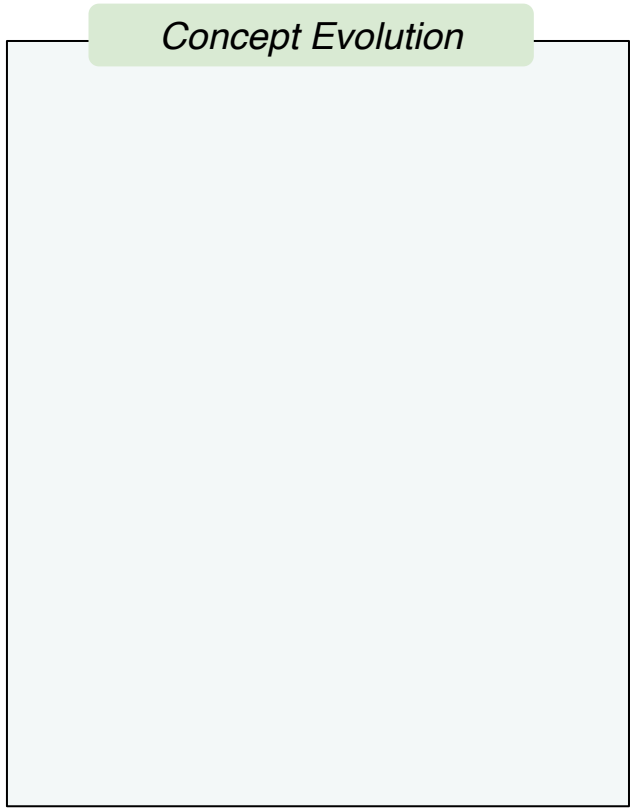


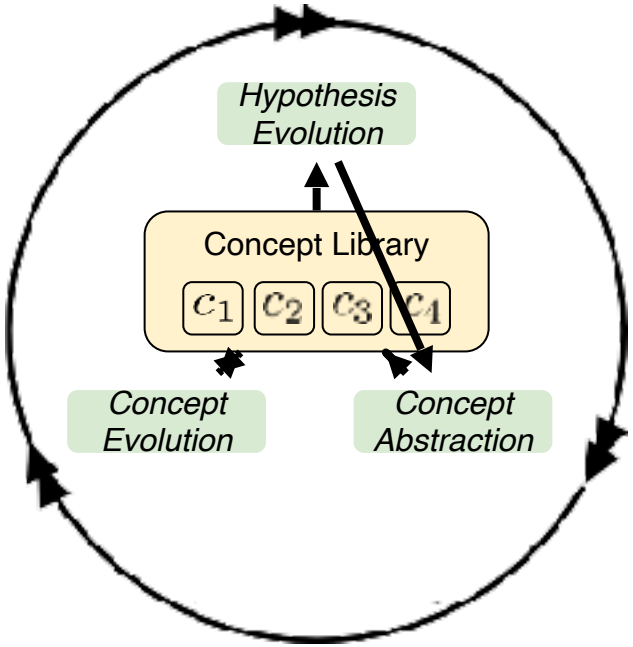
x300 times

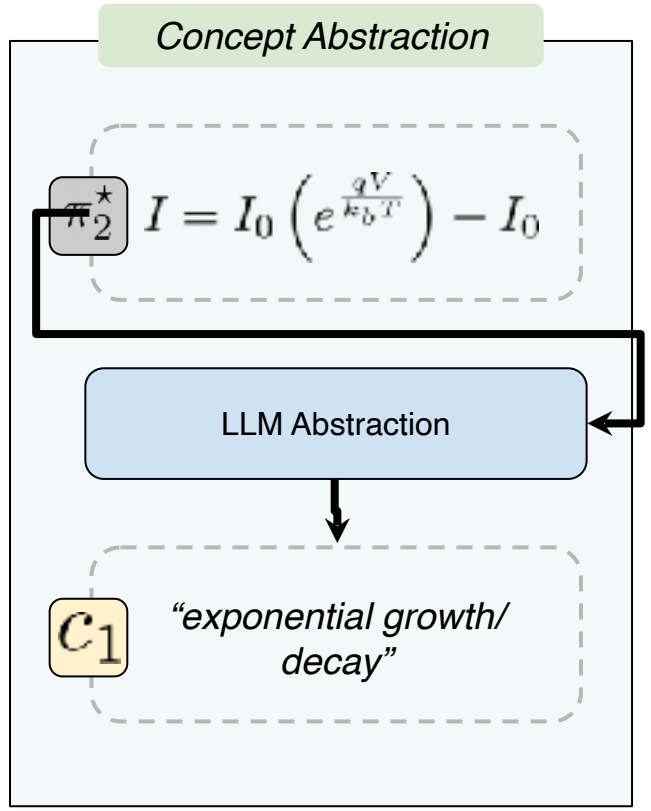
LaSR



LLM Abstraction induces *useful** abstractions.

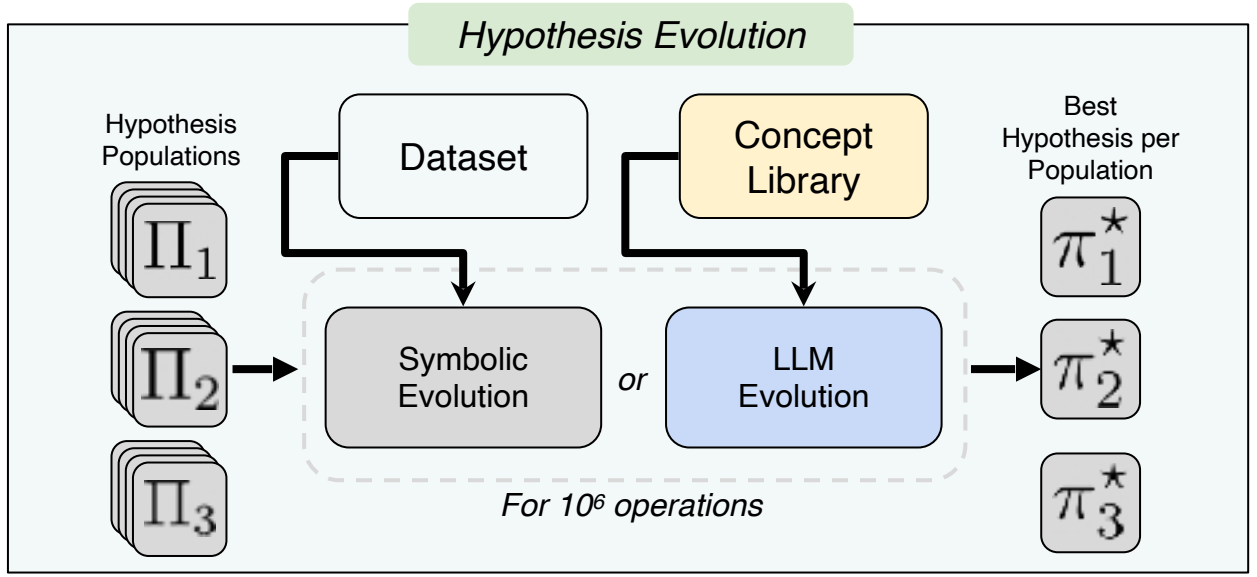


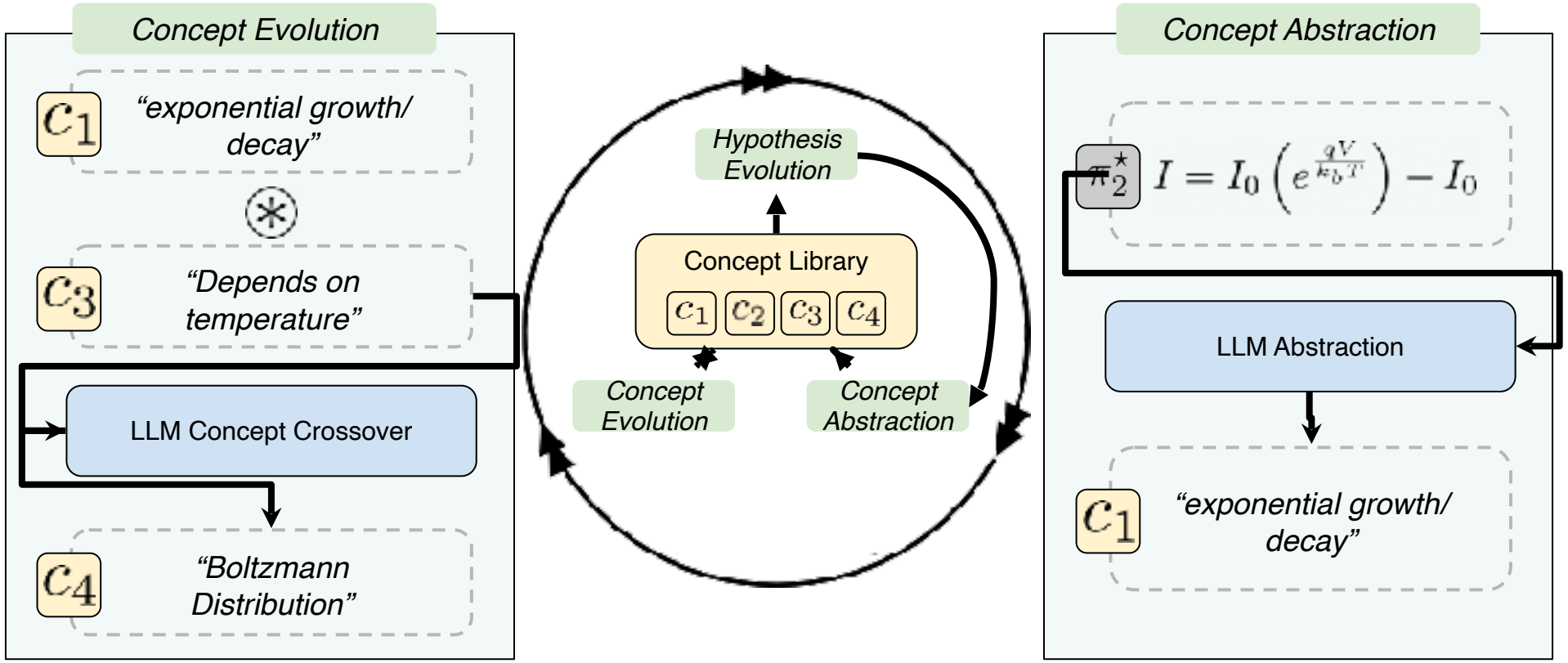




LaSR

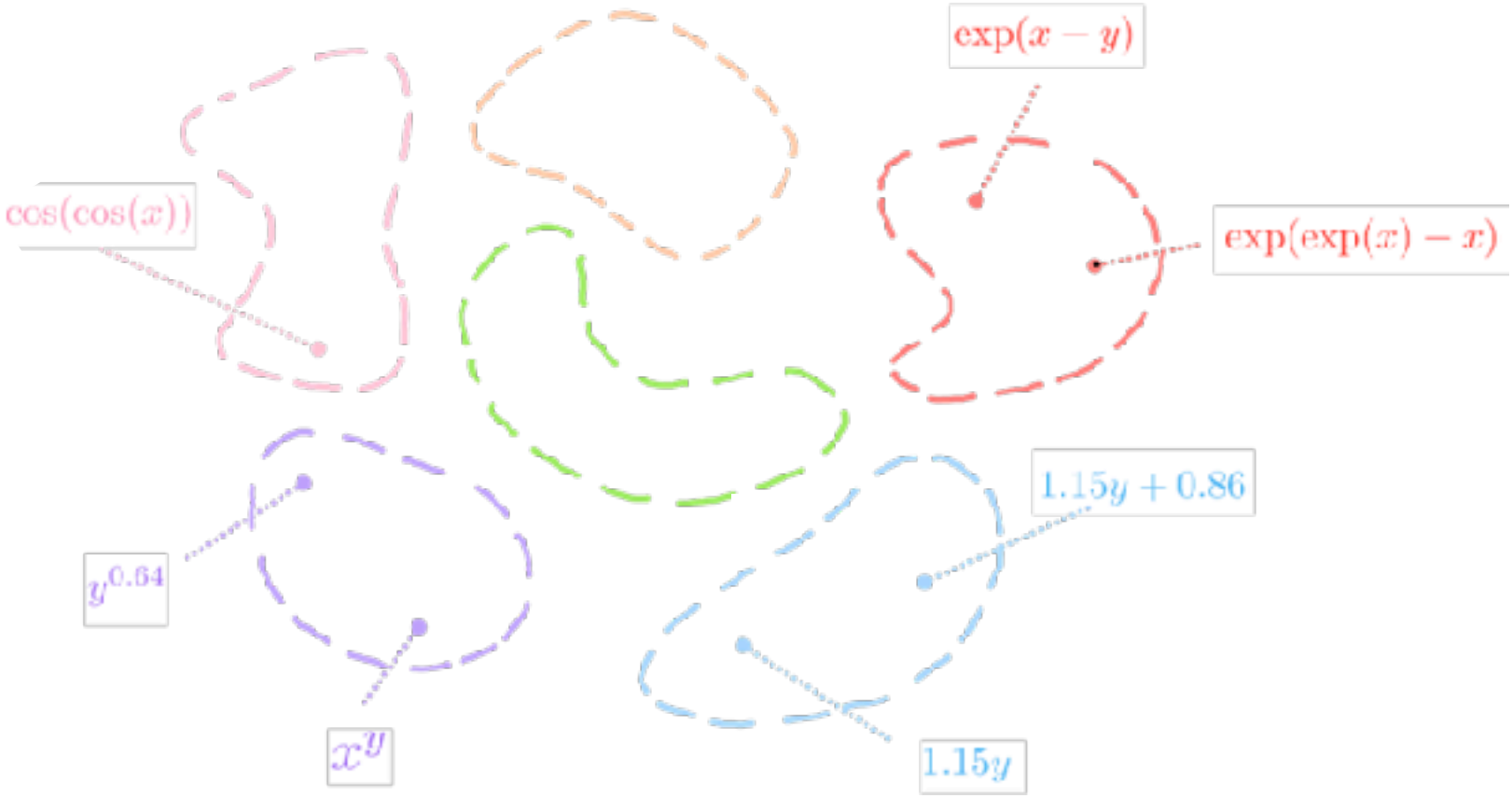
LLM Concept Crossover evolves all concepts.





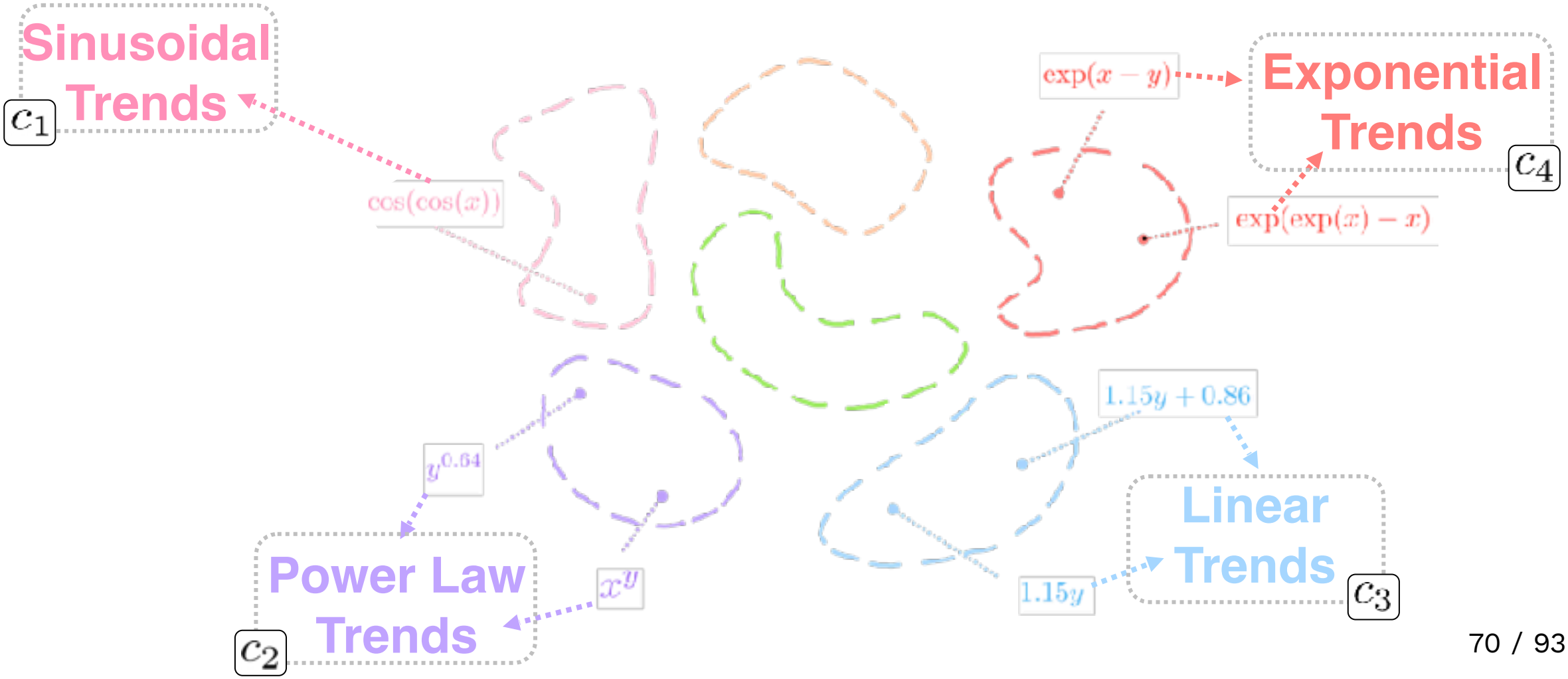
Sketch of Search Space

After Phase 1:
“Islands” of expressions



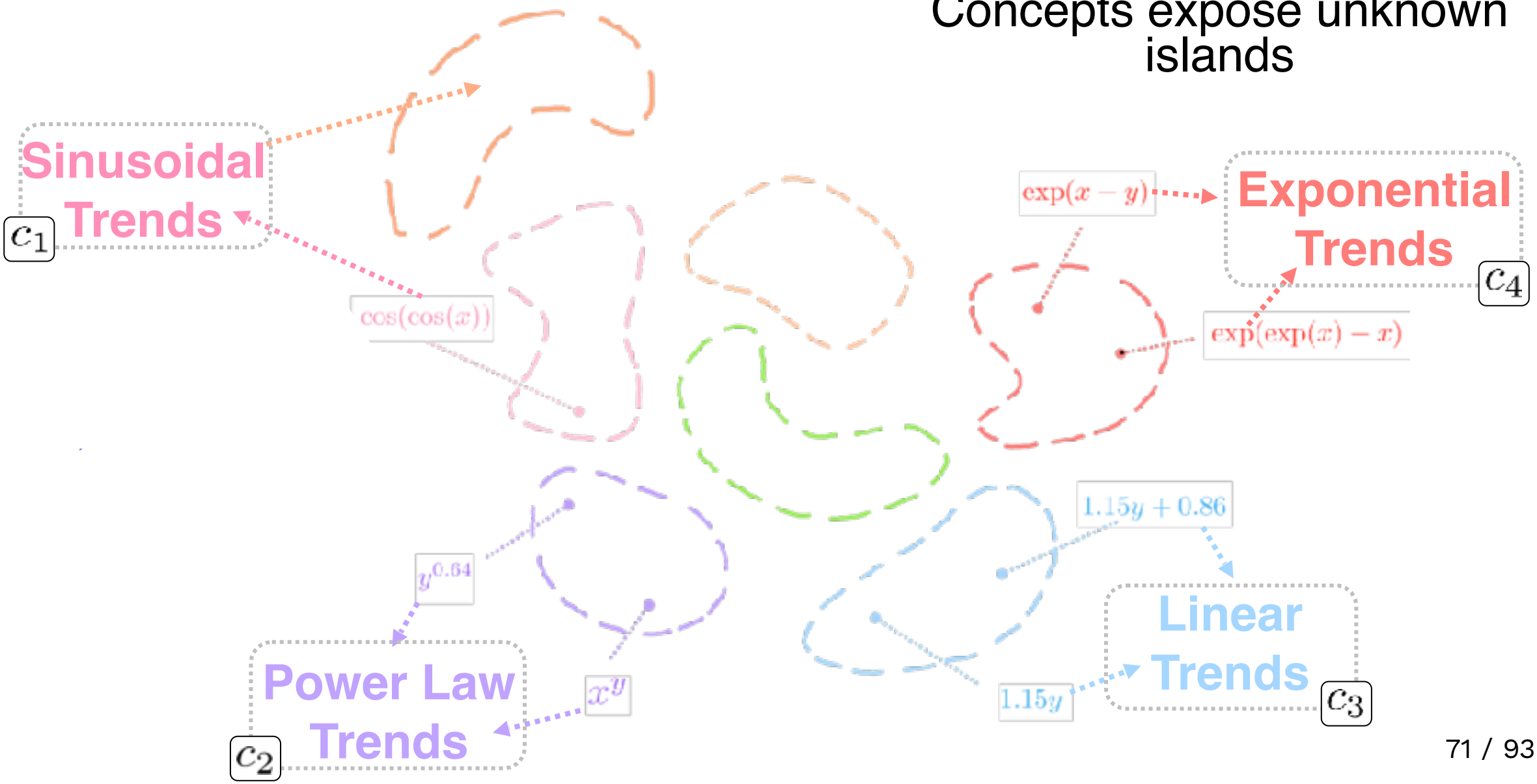
Sketch of Search Space

After Phase 2:
Concepts for each “Island”



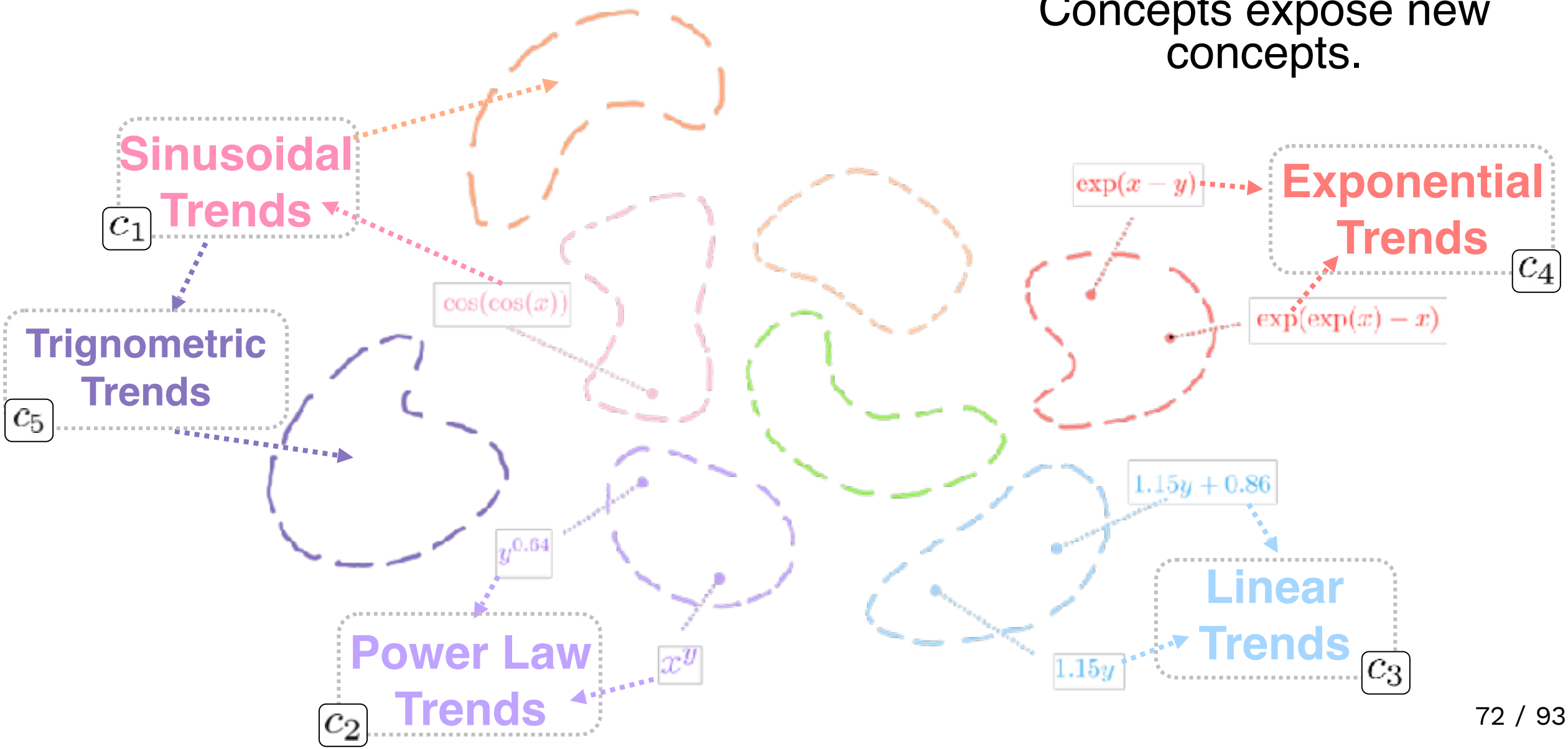
Sketch of Search Space

After Phase 2:
Concepts expose unknown islands



Sketch of Search Space

After Phase 3:
Concepts expose new concepts.



LaSR: Overall Performance

- Concept Guidance accelerates discovery.

GPlearn	AFP	AFP-FE	DSR	uDSR	AIFeynman	PySR	LaSR
20/100	24/100	26/100	23/100	40/100	38/100	59/100	**72/100**

Table 1: Results on 100 Feynman equations from [47]. We report exact match solve rate for all models. LASR achieves the best exact match solve rate using the same hyperparameters as PySR.

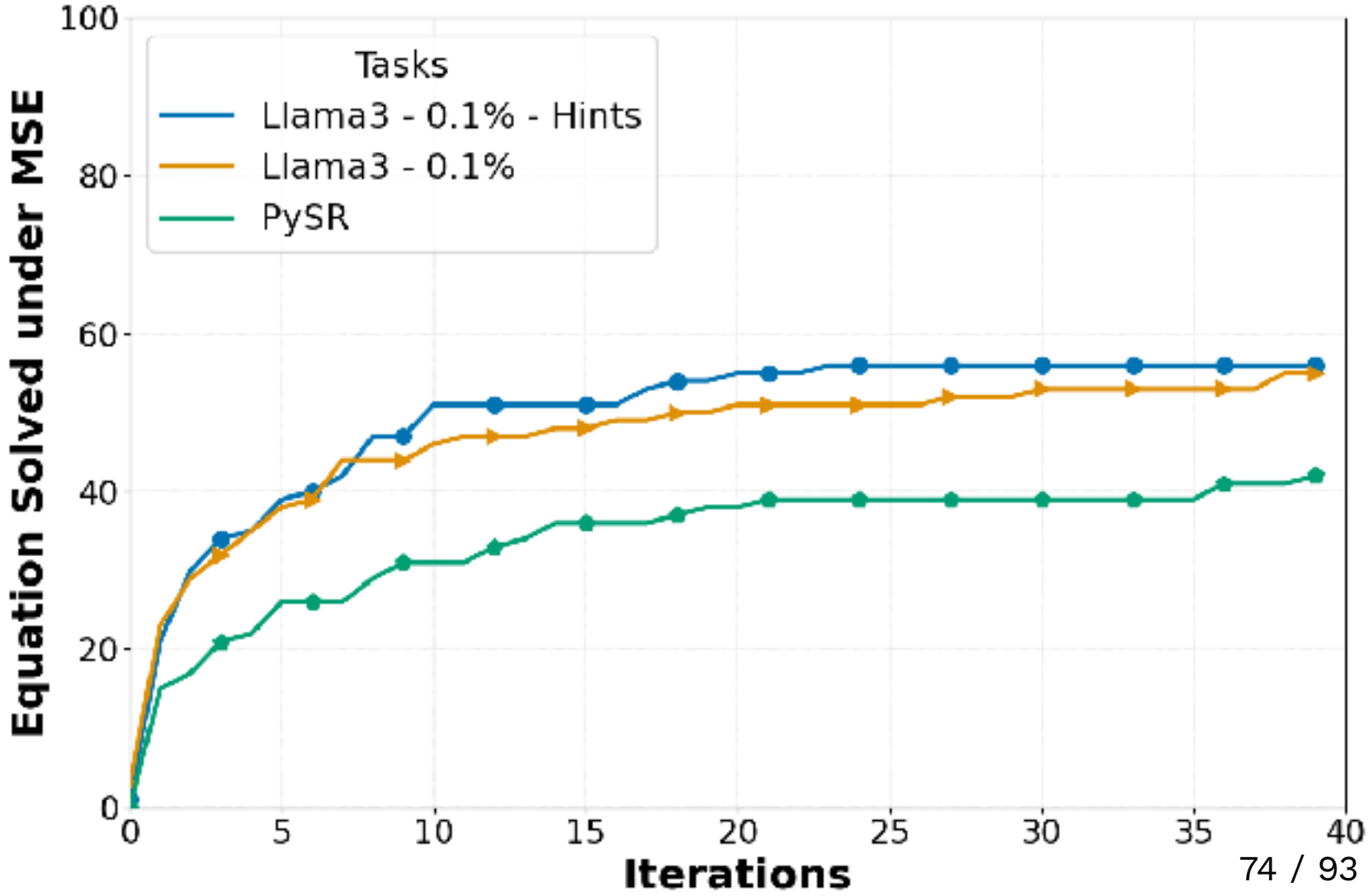
- LaSR outperforms PySR ev with local language models (llama-3-7b, 1%)

		LASR (Llama3-8B)			LASR (GPT-3.5)
Type of Solve	**PySR**	$p = 1\%$	$p = 5\%$	$p = 10\%$	$p = 1\%$
Exact Solve	59/100	67/100	69/100	71/100	72/100
Almost Solve	7/100	5/100	6/100	2/100	3/100
Close	16/100	9/100	12/100	12/100	10/100
Not Close	18/100	19/100	13/100	16/100	15/100

Table 2: Evaluation results on Feynman dataset by cascading LASR's LLM backbone (llama3-8b, gpt-3.5-turbo) and changing the probability of calling the model ($p = [0.01, 0.05, 0.10]$) in the order of increasing concept guidance. LASR outperforms PySR even with minimal concept guidance using an open-source LLM.

Human-provided hints

User-provided hints accelerate hypothesis search



Example: Coulomb's Law

$$F = \frac{1}{4\pi\epsilon}\frac{q_1 q_2}{r^2}$$

$$F = \frac{\left(\left(\left(\left(\left(\left(\frac{q_2 \cdot 3.382}{r}\right) - \left(\frac{\sin\left(\frac{0.017}{\exp(B)}\right)}{\exp(C)}\right)\right)/0.712\right)\cdot q_1\right)\cdot 0.087\right)/\epsilon\right)\cdot 0.191\right)}{r}$$

Eq 10: Coulomb's Law

- Inverse Square Law
- Directly proportional to charges
- Force symmetric w.r.t charges

PySR's Solution

- Reduces to ground truth after 10 steps of simplification.
- Unwieldly
- Fitting more constants => more optimization errors

Coulomb's Law

$$F = \frac{1}{4\pi\epsilon}\frac{q_1 q_2}{r^2}$$

Eq 10: Coulomb's Law

- Inverse Square Law
- Directly proportional to charges
- Force symmetric w.r.t charges

$$\begin{aligned}
F &= \frac{q_1}{\left(\frac{r}{q_2}\right)\left(r + \frac{1.9181636\times10^{-5}}{q_2}\right)\epsilon} \cdot 0.07957782 \\
&= \frac{q_1}{\left(\frac{r}{q_2}\right)\left(r + \frac{1.9181636\times10^{-5}}{q_2}\right)\epsilon} \cdot \frac{1}{4\pi} && \text{(Substitute constant)} \\
&= \frac{q_1 q_2}{r\left(r + \frac{1.9181636\times10^{-5}}{q_2}\right)\epsilon} \cdot \frac{1}{4\pi} && \text{(Simplify denominator)} \\
&\approx \frac{q_1 q_2}{r\,(r)\,\epsilon} \cdot \frac{1}{4\pi} && \text{(Negligible. } \tfrac{1.9181636\times10^{-5}}{q_2} \approx 0\text{)}
\end{aligned}$$

LaSR's Solution

- Reduces to ground truth after 4 steps of simplification
- Smaller models synthesize simpler equations!

Coulomb’s law

$$F = \frac{1}{4\pi\epsilon}\frac{q_1 q_2}{r^2}$$

Eq 10: Coulomb's Law

- Inverse Square Law
- Directly proportional to charges
- Force symmetric w.r.t charges

Iteration	Discovered Concept
2	*The good mathematical expressions exhibit […] with a focus on **power functions and trigonometric functions** […]*
6	*The good mathematical expressions exhibit […] **symmetry or regularity** […]*
24	*The good mathematical expressions have […] with a specific pattern of **division and multiplication***

LaSR’s Concepts (Limitations)

- Cannot guarantee factuality or correctness.
- Good concepts depend on LLM training. Concepts can mislead scientists.

Finding LLM Scaling Laws

Step 1: Postulate Scaling Law

Step 2: Measure model loss w.r.t hyper parameters.

Step 3: Fit scaling law to dataset.

$$L(N, D) = \underbrace{\frac{A}{N^{\alpha}}}_{\text{finite model}} + \underbrace{\frac{B}{D^{\beta}}}_{\text{finite data}} + \underbrace{E}_{\text{irreducible}}$$

Google DeepMind MassiveText

$$L(N, D) = \underbrace{\frac{406.4}{N^{0.34}}}_{\text{finite model}} + \underbrace{\frac{410.7}{D^{0.28}}}_{\text{finite data}} + \underbrace{1.69}_{\text{irreducible}}$$

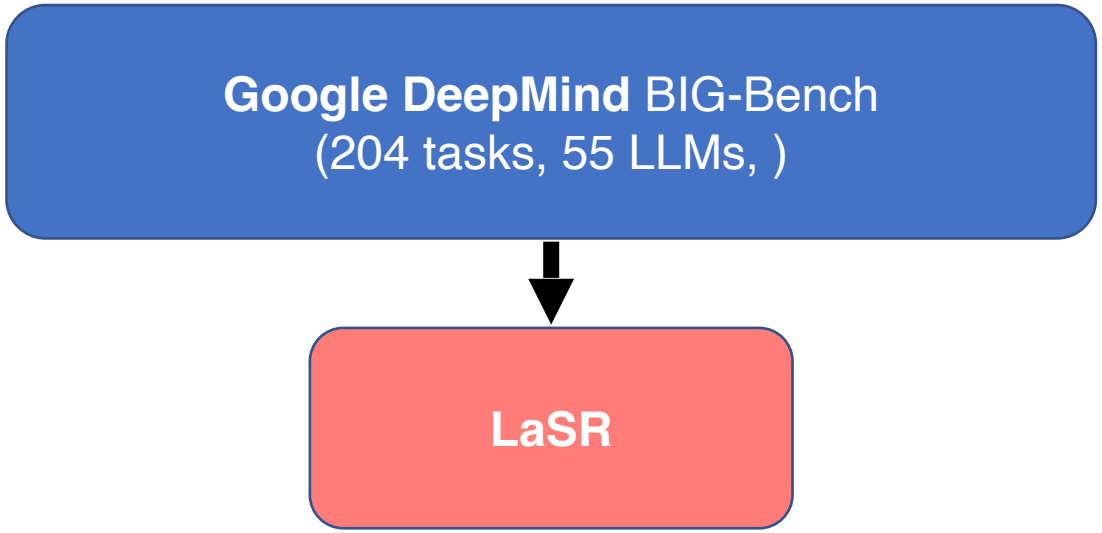
Training Compute-Optimal Large Language Models, Hoffman et. al

Finding LLM Scaling Laws with LaSR

Step 1: Measure model loss w.r.t hyper parameters.

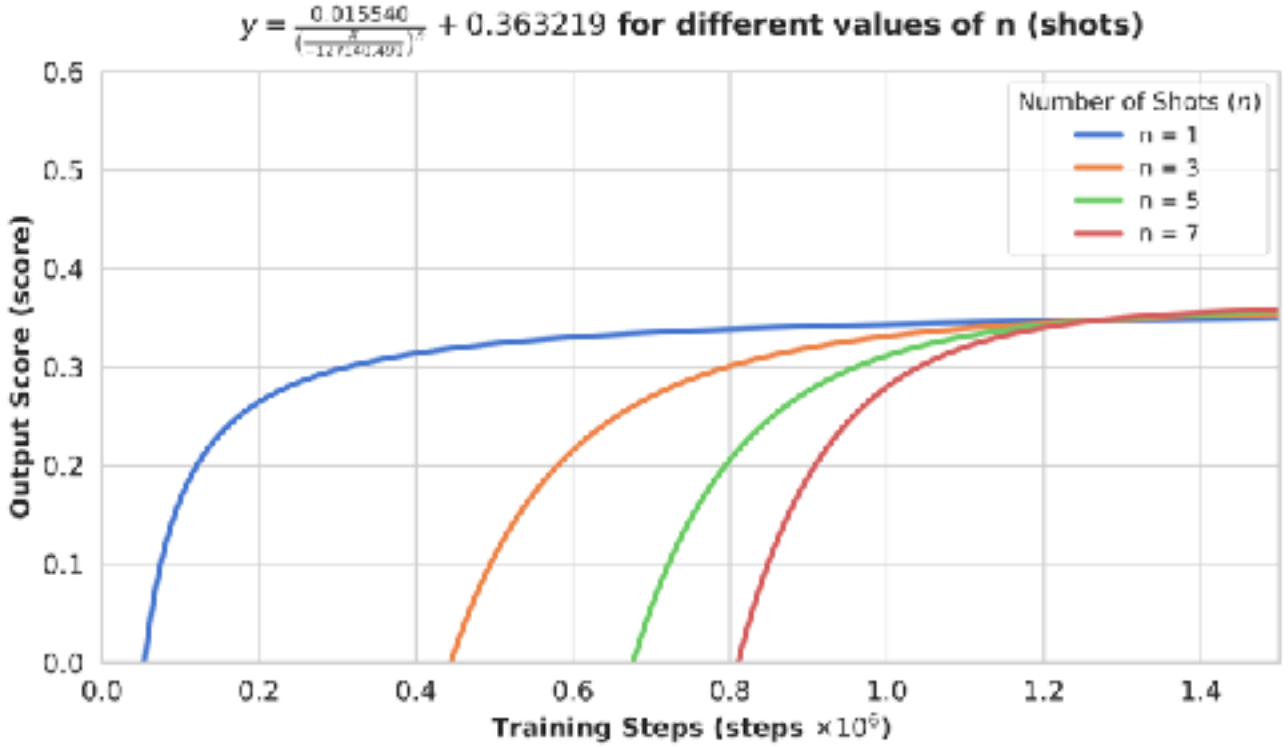
Step 2: Use symbolic regression to postulate and fit scaling laws

Step 3: Choose the scaling law that fits the data the best while using the least parameters.



Training Compute-Optimal Large Language Models, Hoffman et. al

LLM Scaling Law



- A large number of shots gives poor results for low-capability models
- Once the models pass a capability threshold, having more shots helps

Qualitative Traits

- Interaction between training hyper params (#steps) and testing hyper params (#shots)

$$\texttt{score} = \frac{A}{\left(\frac{\texttt{train\_steps}}{B}\right)^{\texttt{\#shots}}} + E$$

- We can modify Chinchilla using empirical insights

$$\texttt{score} = \frac{A}{(\texttt{train\_steps} \cdot \texttt{batch\_size})^{\alpha}} + \frac{B}{(\texttt{\#params})^{\beta}} + E \qquad \text{(Chinchilla [22])}$$

$$\texttt{score} = \frac{A}{(\texttt{train\_steps} \cdot \texttt{batch\_size})^{\alpha \cdot \texttt{\#shots}}} + \frac{B}{(\texttt{\#params})^{\beta}} + E \qquad \text{(Modified Chinchilla)}$$

Scaling Law Skeleton	MSE Loss	Free Parameters
Equation 4	0.03655 ± 0.00281	3
Chinchilla [22]	0.03664 ± 0.00283	5
Modified Chinchilla	$\mathbf{0.03655 \pm 0.00280}$	5
Residual Term Only	0.09324 ± 0.01992	1

Self-Evolving Visual Concept Library Using Vision-Language Critics

Atharva Sehgal, Patrick Yuan, Ziniu Hu, Yisong Yue, Jennifer J Sun, Swarat Chaudhuri.
Computer Vision and Pattern Recognition, 2025.

Atharva

Zero-shot transfer learning with VLMs

Identify the adult bald eagle

?

Identify the ever given.

?

[Menon & Vondrick, ICLR 2023]

Classification with visual concept descriptors

Identify the adult bald eagle

LLM

Large wings
Large beak
White head
Black body

VLM

Identify the ever given.

LLM

Container ship
Evergreen
Stacked containers

VLM

[Menon & Vondrick, ICLR 2023]

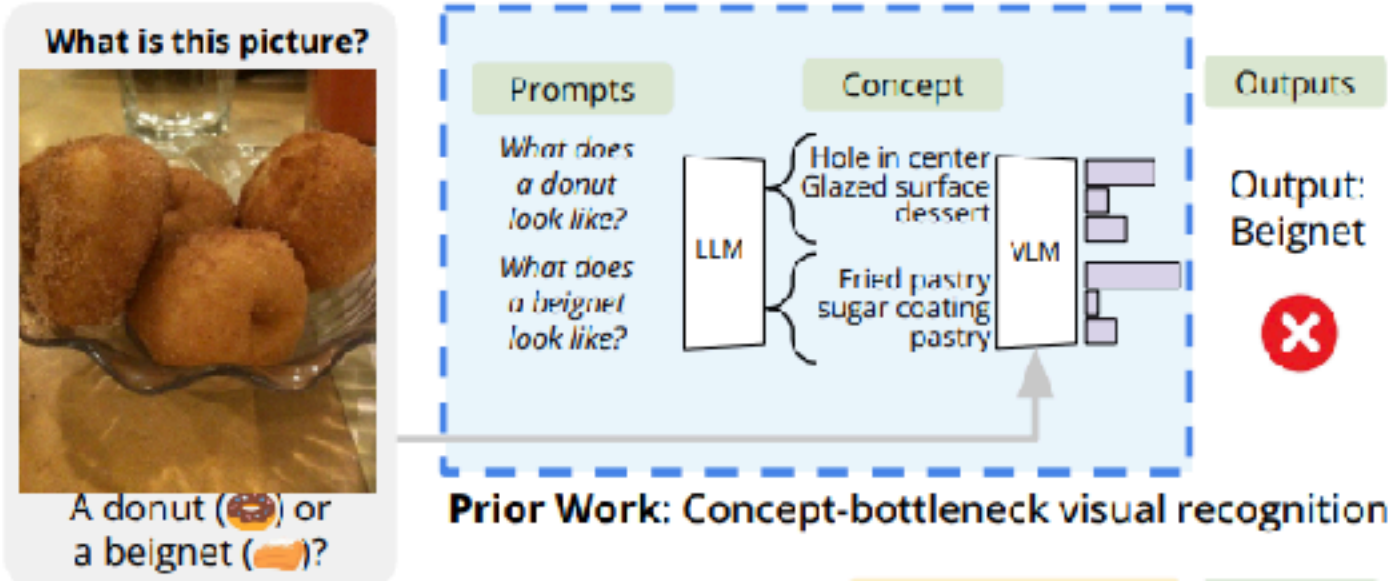
Concept descriptors can be used to write code

Is this an image of an adult bald eagle?

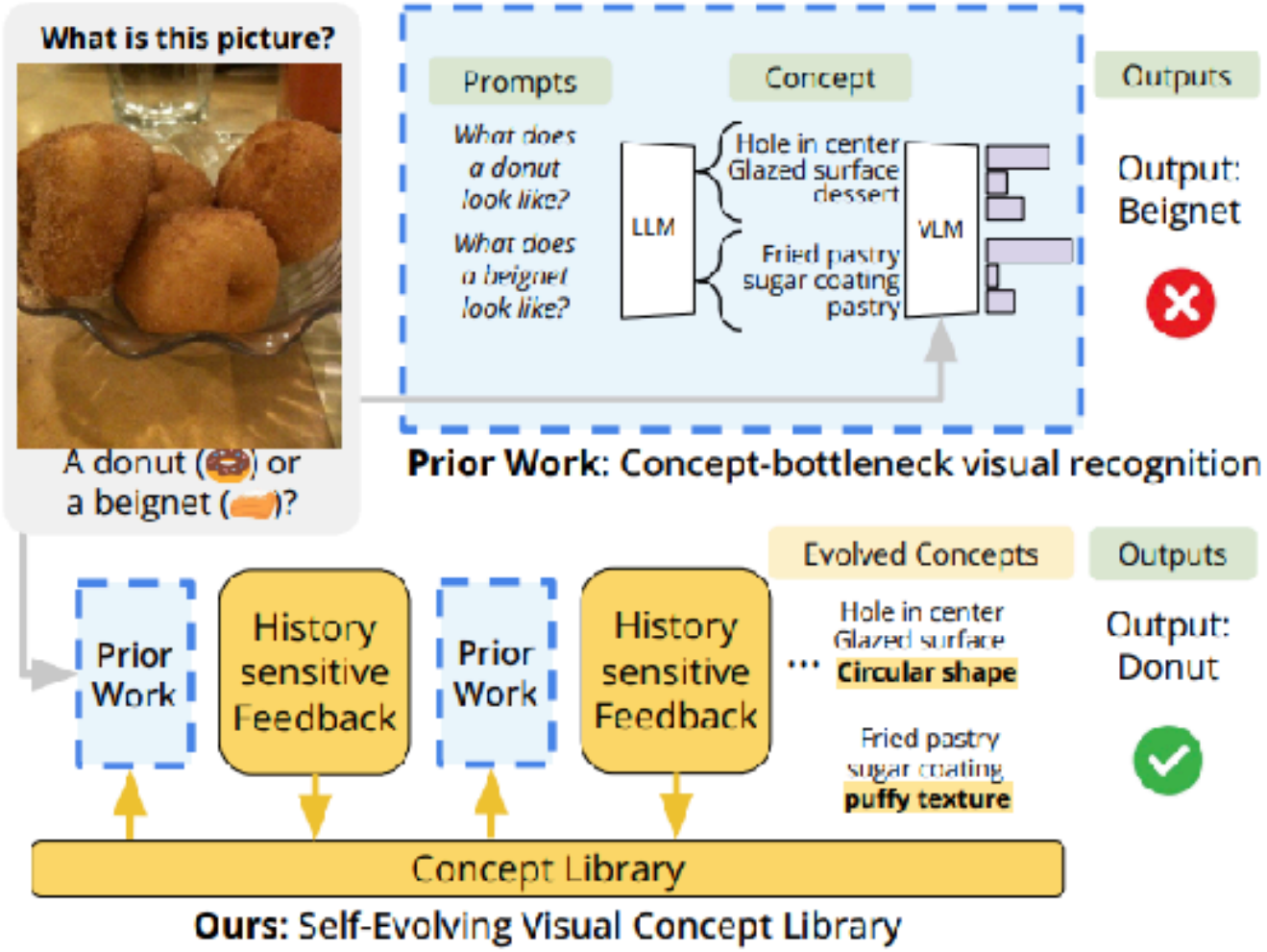
GPT

```
def is_eagle(image):
    image_patch = ImagePatch(image)
    if not image_patch.exists('bird'):
        return 'no'
    eagle_patch = image_patch.find('bird')[0]
    eagle_features = [
        "large wings",
        "large beak",
        "white head",
        "black body"
        ]
    p_eagle = 0.0
    for feature in eagle_features:
        p_eagle += eagle_patch.exists(feature)
    p_eagle /= len(eagle_features)
    return 'yes' if p_eagle >= 0.75 else 'no'
```

Concept Refinement with Evolution



Concept Refinement with Evolution

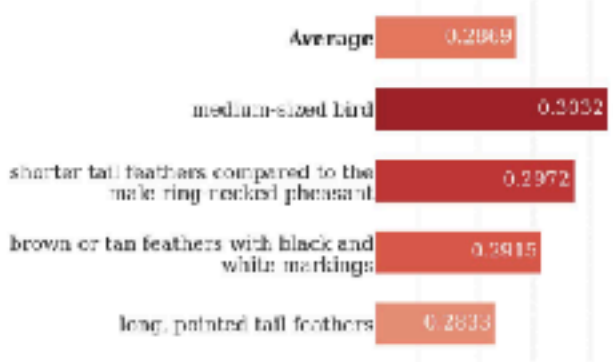


- VLM evaluates the similarity between each image and its associated concepts compared to other images, and computes a contrastive score.
- This score is used to refine the library of visual concepts.

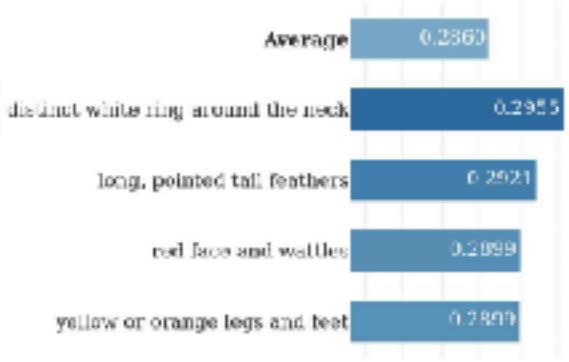
Sample Result

This is a **Male Ring-necked pheasant.**

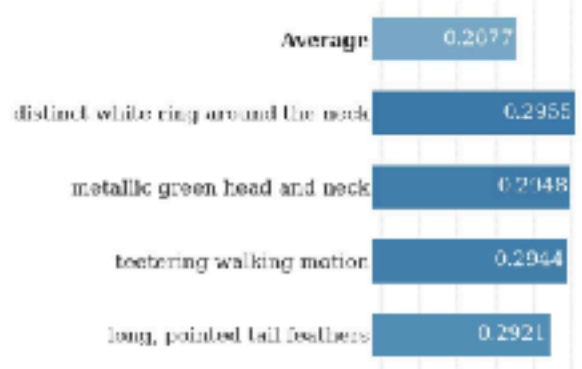
With no iterations, the baseline confuses this for an **Female Ring-necked pheasant** because:



While the **true class** has lower aggregate activation because:



After iteration with ESCHER, the baseline correctly predicts this to be a **Male Ring-necked pheasant** because:



Summary: LLM Agents for Empirical Discovery

LLM-directed evolution is a powerful tool for empirical scientific discovery.

Frontier LLMs inject prior world knowledge into mutation/crossover operators.

LLMs can be used to learn abstract concepts that accelerate evolution.

All this can be applied to settings with visual inputs as well.

Open Challenges

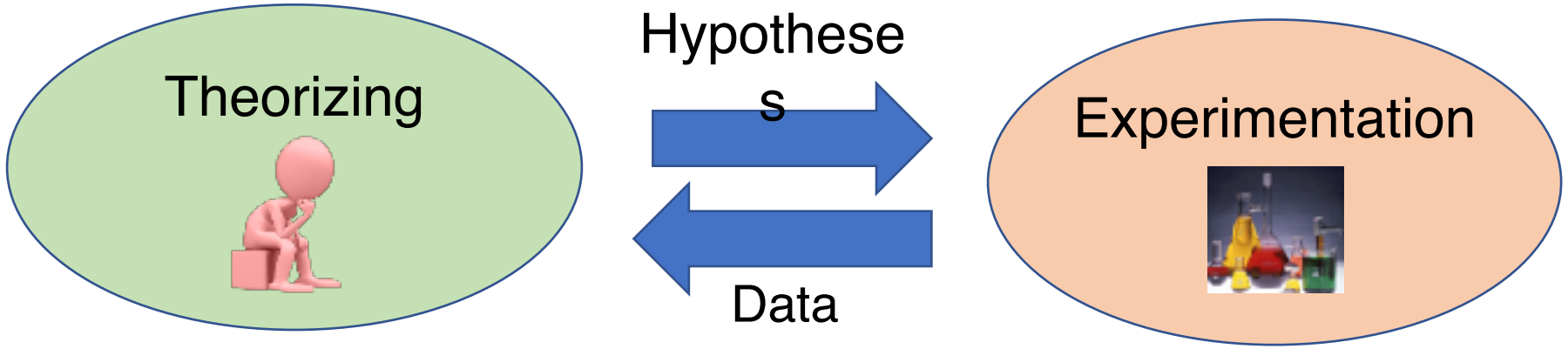
Hypothesis and concept **verification**

Concept representations beyond natural language

Scaling to larger search spaces and input dimensions

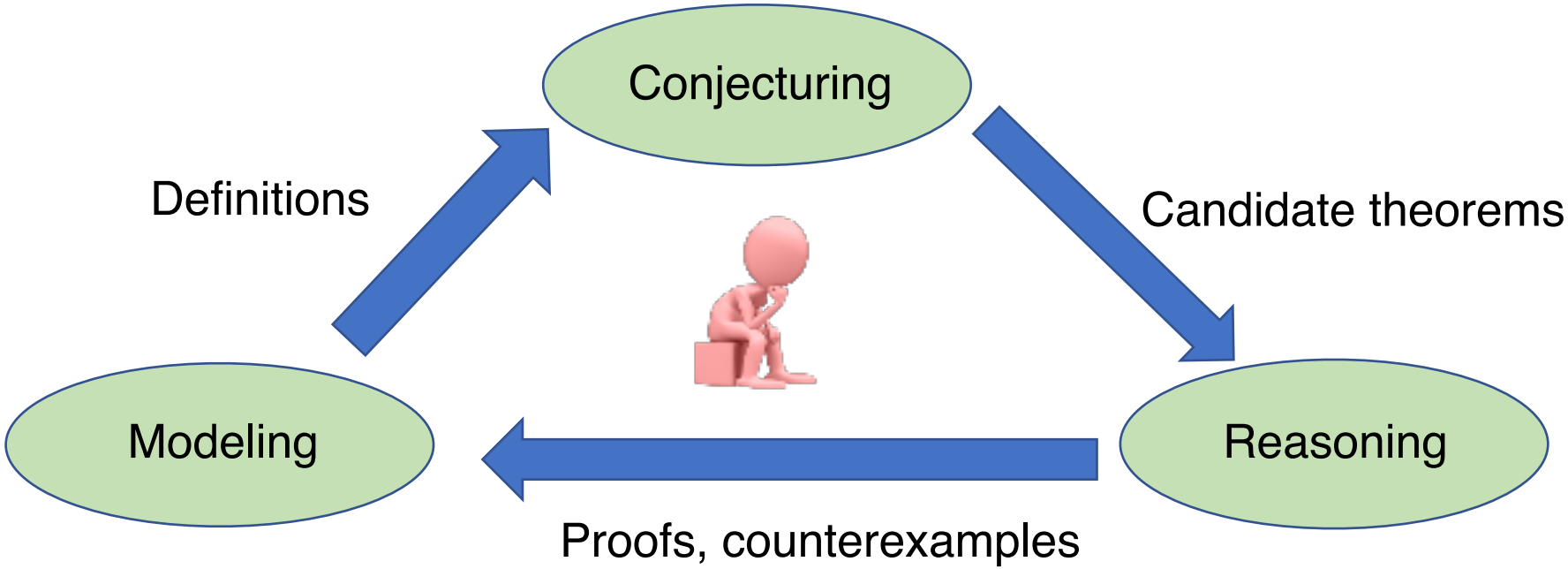
Going beyond hypothesis generation to **experiment design**

Scientific discovery



LLM agents are extraordinarily powerful tools for scientific discovery.

Mathematical discovery



AI for math: Automate conjecturing and proof

Collaborators and Funders