Program Induction & Skill Learning

Sasha Boguraev and Dori Litvak

High Level Overview of Program Induction

- Premise:
  - Represent knowledge as programs (symbolic code)
  - Learning is adding new knowledge, or in this case, making new programs (program synthesis)
- Why Program Induction?
  - Programs are generalizable, data efficient, and interpretable

Slide adapted from Kevin Ellis talk at the Simons Institute (2021)

Basic Methodological Background

- **Formulation:** Given a set of inputs and corresponding outputs, induce a program for which running it on a given input will result in the corresponding output
- Typical Approach (pre-LLM era):
  - Define a DSL (Domain Specific Language)
  - Sample programs according to some probabilistic rules
  - Update your sampling rules according to your success rate
- Usually done in a Bayesian manner (define a prior over a DSL and sample using monte-carlo methods updating your prior)

Examples

Flashfill (Gulwani 2012)

EXAMPLE 5 (Split Odds). *The goal in this problem, taken from an Excel help forum, is to place each odd in a separate cell, while ignoring any extraneous numbers or parenthesis. We reduce the problem of generating multiple unbounded number of output strings to that of generating one output string where the multiple strings are separated by a unique symbol, say #.*

Input v_1	*Output*
(6/7)(4/5)(14/1)	*6/7 # 4/5 # 14/1 #*
49(28/11)(14/1)	*28/11 # 14/1 #*
() (28/11)(14/1)	*28/11 # 14/1 #*

String Program:

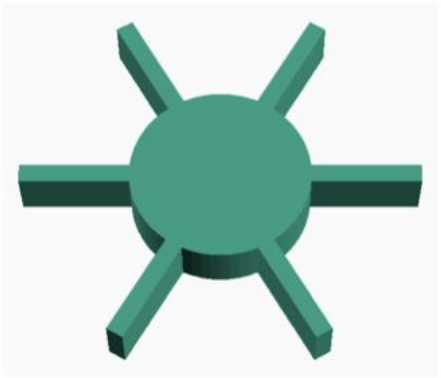
$\texttt{Loop}(\lambda w : \texttt{Concatenate}(\texttt{SubStr}(v_1, p_1, p_2), \texttt{ConstStr}(\text{“\# ”})))$
where $p_1 \equiv \texttt{Pos}(\textit{LeftParenTok}, \texttt{TokenSeq}(\textit{NumTok}, \textit{SlashTok}), w))$
and $p_2 \equiv \texttt{Pos}(\texttt{TokenSeq}(\textit{SlashTok}, \textit{NumTok}), \textit{RightParenTok}, w)$.

EXAMPLE 6 (Remove excess spaces). *The goal in this problem, provided by the product team and also present in [21], is to remove all leading and trailing spaces and replace internal strings of multiple spaces by a single space. Notice how the loop expression prints out all but last sequence of non-whitespace characters (to not print any trailing whitespace in the output).*

Input v_1	*Output*
Oege de Moor	*Oege de Moor*
Kathleen Fisher AT&T Labs	*Kathleen Fisher AT&T Labs*

String Program:

$\texttt{Concatenate}(\texttt{Loop}(\lambda w : \texttt{Concatenate}(\texttt{SubStr}(v_1, p_1, p_2)), \texttt{ConstStr}(\text{“ ”})), \texttt{SubStr2}(v_1, \textit{NonSpaceTok}, -1))$
where $p_1 \equiv \texttt{Pos}(\epsilon, \textit{NonSpaceTok}, w)$, *and*
$p_2 \equiv \texttt{Pos}(\textit{NonSpaceTok}, \texttt{TokenSeq}(\textit{SpaceTok}, \textit{NonSpaceTok}), w)$.



(a) CAD model of ship's wheel

```
(Union
 (Cylinder [1, 5, 5])
 (Fold Union
  (Tabulate (i 6)
   (Rotate [0, 0, 60i]
    (Translate [1, −0.5, 0]
     (Cuboid [10, 1, 1])))))))
```

(b) Caddy program

Szalinski (Nandi et al. 2020)

Examples

Flashfill (Gulwani 2012)

$$
\begin{array}{rcl}
\text{String expr } P & := & \texttt{Switch}((\mathrm{b}_1, \mathrm{e}_1), \cdot\cdot, (\mathrm{b}_n, \mathrm{e}_n)) \\
\text{Bool b} & := & \mathrm{d}_1 \vee \cdot\cdot \vee \mathrm{d}_n \\
\text{Conjunct d} & := & \pi_1 \wedge \cdot\cdot \wedge \pi_n \\
\text{Predicate } \pi & := & \texttt{Match}(v_i, \mathrm{r}, k) \mid \neg\, \texttt{Match}(v_i, \mathrm{r}, k) \\
\text{Trace expr e} & := & \texttt{Concatenate}(\mathrm{f}_1, \cdot\cdot, \mathrm{f}_n) \\
\text{Atomic expr f} & := & \texttt{SubStr}(v_i, \mathrm{p}_1, \mathrm{p}_2) \\
 & \mid & \texttt{ConstStr}(s) \\
 & \mid & \texttt{Loop}(\lambda w : \mathrm{e}) \\
\text{Position p} & := & \texttt{CPos}(k) \mid \texttt{Pos}(\mathrm{r}_1, \mathrm{r}_2, \mathrm{c}) \\
\text{Integer expr c} & := & k \mid k_1 w + k_2 \\
\text{Regular Expression r} & := & \texttt{TokenSeq}(\mathrm{T}_1, \cdot\cdot, \mathrm{T}_m) \\
\text{Token T} & := & C+ \mid [\neg C]+ \\
 & \mid & \text{SpecialToken}
\end{array}
$$

Figure 1. Syntax of String Expressions P. v_i refers to a free string variable, while w refers to a bound integer variable. k denotes an integer constant and s denotes a string constant.

$$
\begin{array}{lcl}
\text{op} & ::= & + \mid - \mid \times \mid / \\
\text{num} & ::= & \mathbb{R} \mid \langle var \rangle \mid \langle num \rangle\ \langle op \rangle\ \langle num \rangle \\
\text{vec2} & ::= & [\langle num \rangle, \langle num \rangle] \\
\text{vec3} & ::= & [\langle num \rangle, \langle num \rangle, \langle num \rangle] \\
\text{affine} & ::= & \text{Translate} \mid \text{Rotate} \mid \text{Scale} \mid \textbf{TranslateSpherical} \\
\text{binop} & ::= & \text{Union} \mid \text{Difference} \mid \text{Intersection} \\
\text{cad} & ::= & (\text{Cuboid } \langle vec3 \rangle) \mid (\text{Sphere } \langle num \rangle) \\
 & \mid & (\text{Cylinder } \langle vec2 \rangle) \mid (\text{HexPrism } \langle vec2 \rangle) \mid \ldots \\
 & \mid & (\langle affine \rangle\ \langle vec3 \rangle\ \langle cad \rangle) \\
 & \mid & (\langle binop \rangle\ \langle cad \rangle\ \langle cad \rangle) \\
 & \mid & (\textbf{Fold}\ \langle binop \rangle\ \langle cad\text{-}list \rangle) \\
\text{cad-list} & ::= & (\text{List } \langle cad \rangle +) \\
 & \mid & (\text{Concat } \langle cad\text{-}list \rangle +) \\
 & \mid & (\text{Tabulate } (\langle var \rangle\ \ \mathbb{Z}^+) +\ \langle cad \rangle) \\
 & \mid & (\text{Map2 } \langle affine \rangle\ \langle vec3\text{-}list \rangle\ \langle cad\text{-}list \rangle) \\
\text{vec3-list} & ::= & (\text{List } \langle vec3 \rangle +) \\
 & \mid & (\text{Concat } \langle vec3\text{-}list \rangle +) \\
 & \mid & (\text{Tabulate } (\langle var \rangle\ \ \mathbb{Z}^+) +\ \langle vec3 \rangle)
\end{array}
$$

Figure 2. Caddy syntax. The Core Caddy (CSG) subset omits variables, list forms (those using **Fold**), and **TranslateSpherical**.

Szalinski (Nandi et al. 2020)

Issues

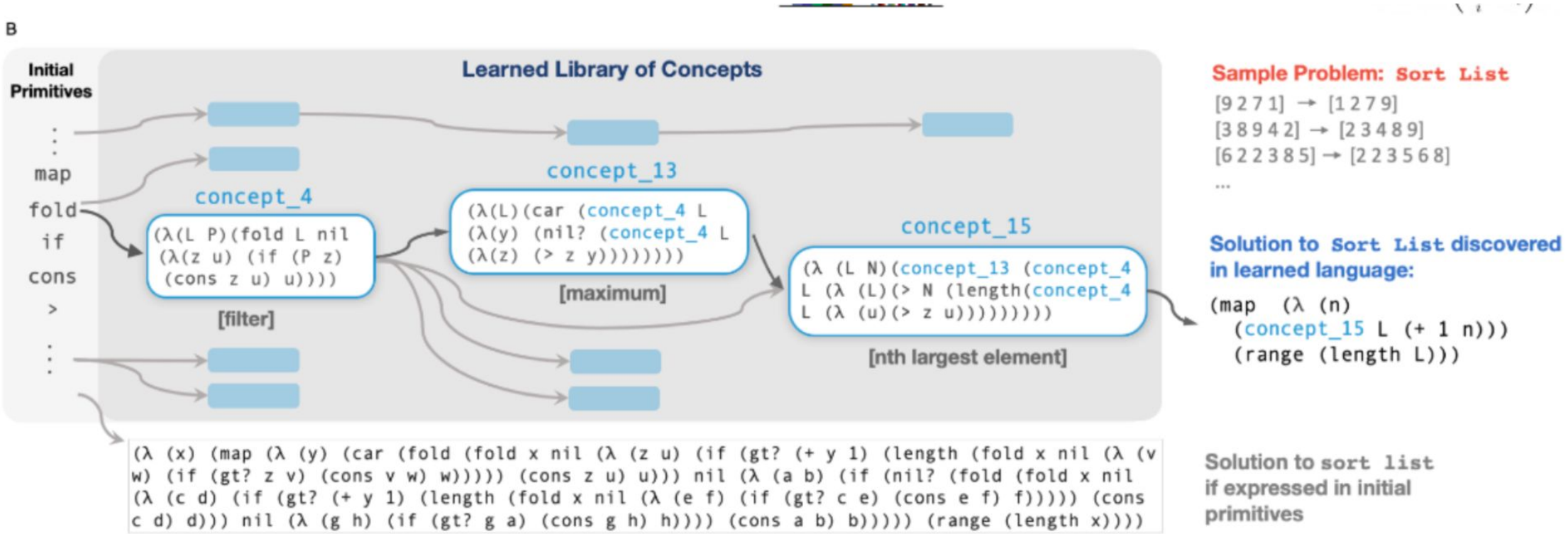
1. Typically requires a carefully crafted DSL + accompanying prior (previous slides)

2. Even with a nice DSL, finding general-use algorithms is intractable due to the combinatorial nature of the search space

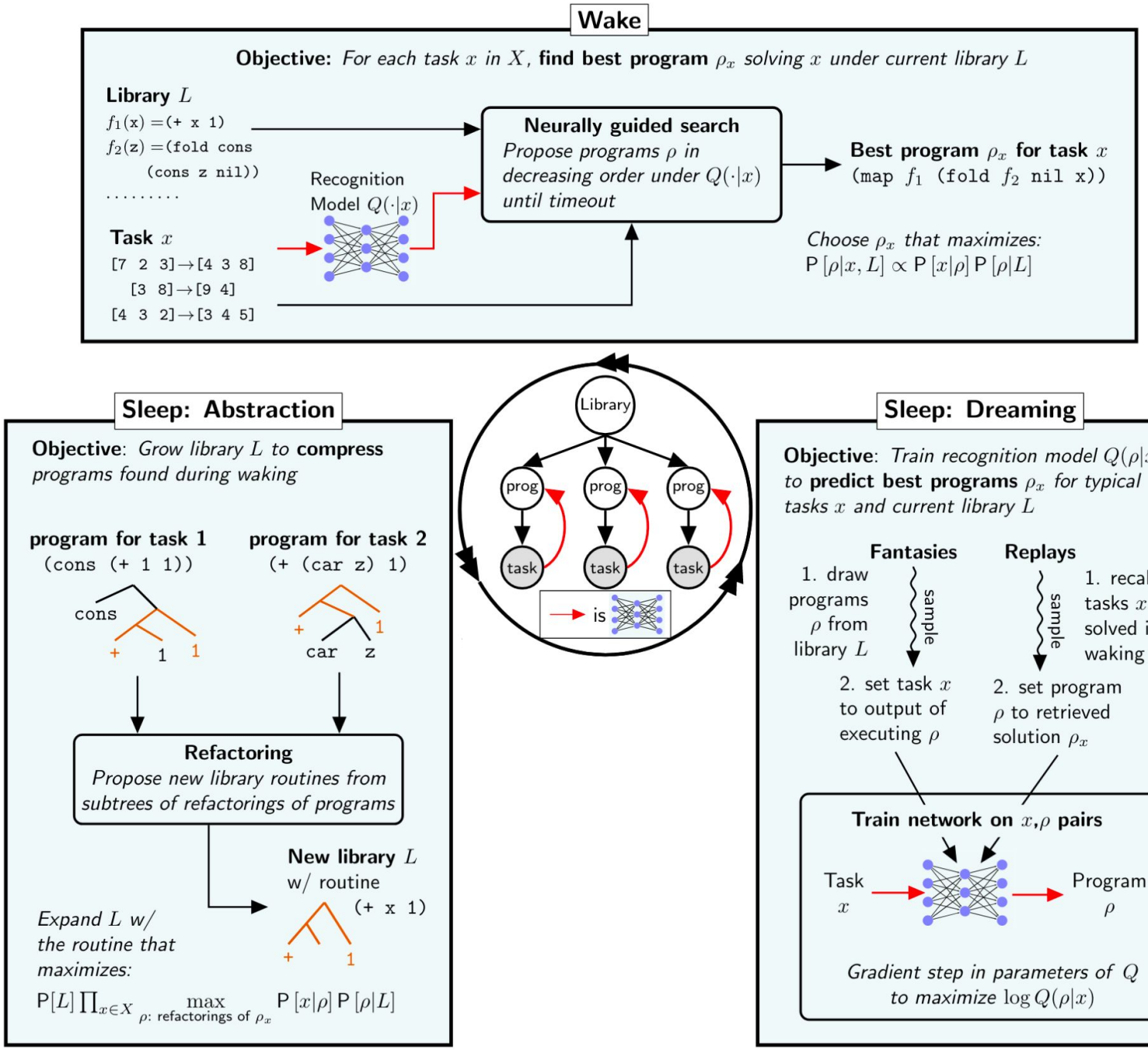
DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning

Ellis et al.

High Level Overview



- Full process: Less than 10 minutes of search
- For reference, the full solution would take in excess of 10^{72} years to brute force

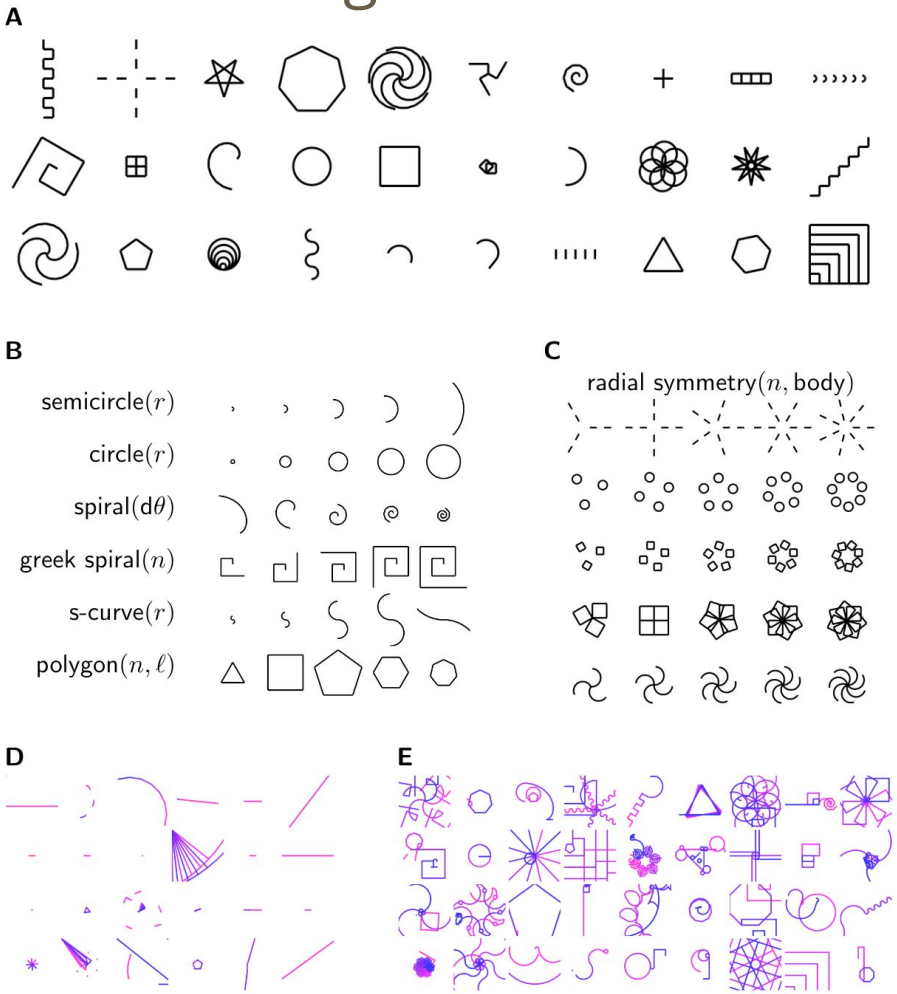


Note: Train until timeout or until task completed

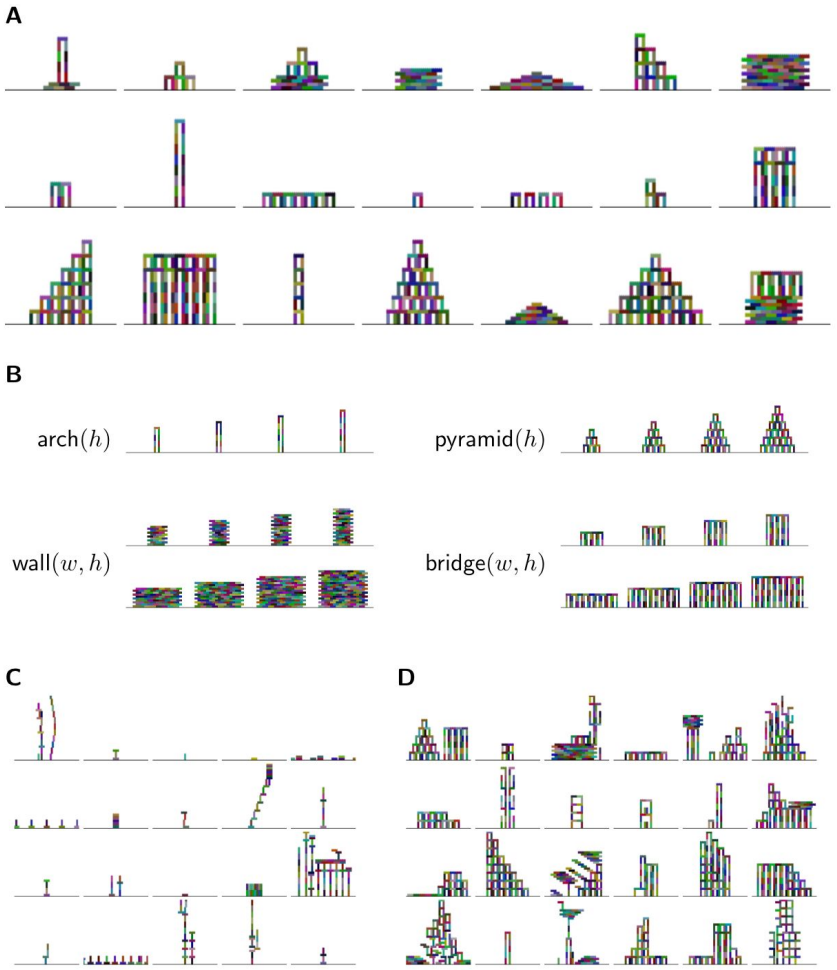
Note: 'Fantasies' (random programs) are used to generate excess training data

Evaluation + Results

Logo Task

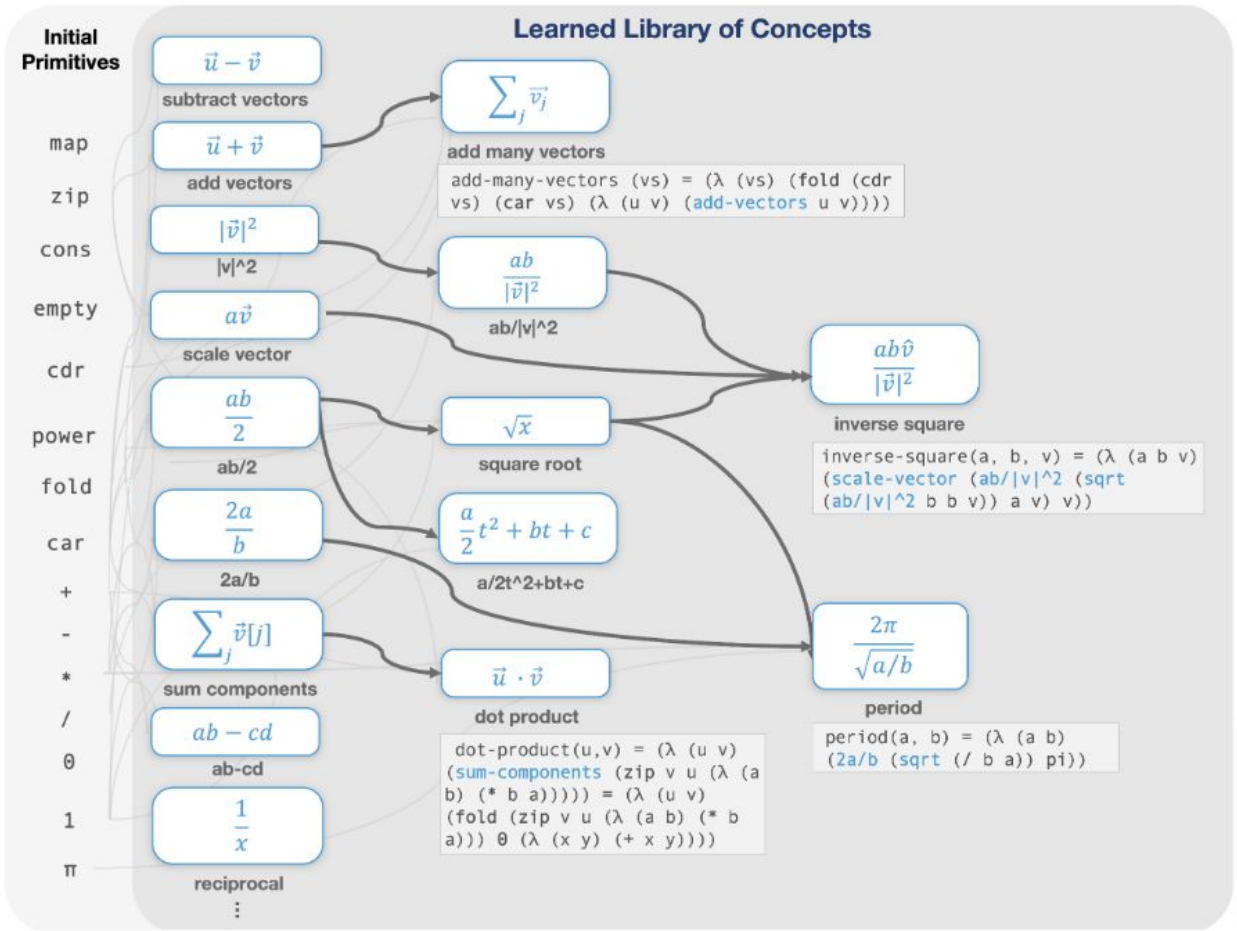


Tower Building Task



Evaluation + Results

A



Discovered Physics Equations

Newton's Second Law

$$\vec{a} = \frac{1}{m}\sum_i \vec{F}_i$$

```
(scale-vector(reciprocal m)
(add-many-vectors Fs))
```

Parallel Resistors

$$R_{total} = \left(\Sigma_i \frac{1}{R_i}\right)^{-1}$$

```
(reciprocal (sum-components
(map (λ(r) (reciprocal r))
Rs)))
```

Work

$$U = \vec{F} \cdot \vec{d}$$

```
(dot-product F d)
```

Force in a Magnetic Field

$$|\vec{F}| = q|\vec{v} \times \vec{B}|$$

```
(* q (ab-cd v_x b_y v_y b_x))
```

Kinetic Energy

$$KE = \frac{1}{2}m|\vec{v}|^2$$

```
(ab/2 m (|v|^2 v))
```

Coulomb's Law

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r_1} - \vec{r_2}|^2}\widehat{r_1 - r_2}$$

```
(inverse-square q_1 q_2
(subtract-vectors r_1 r_2))
```

```
(λ (x y z u) (map (λ (v) (* (/
(* (power (/ (* x x) (fold (zip
z u (λ (w a) (- w a))) 0 (λ (b
c) (+ (* b b)    c)))) (/ (* 1
1) (+ 1 1))) y) (fold (zip z u
(λ (d e)  (- d e))) 0 (λ (f g)
(+ (* f f) g)))) v)) (zip z u
(λ (h i) (- h i)))))
```

Solution to Coulomb's Law if expressed in initial primitives

Key Insights and Enabling Ideas (tl;dr)

- Jointly learning programs and libraries enables the rapid learning of general, large-scale programs
- Neurally guided synthesis is also key to making search over the combinatorial large program space tractable
- However, agents **do not have world-knowledge**
- Questions?

Voyager: An Open-Ended Embodied Agent with Large Language Models

Wang et al.

Motivation

- With the advent of LLMs one can now develop LLM based agents, harnessing the world-knowledge garnered through pre-training to develop **action plans** or **executable policies**
- However, LLMs are **not** lifelong learners which can acquire, update, accumulate and transfer knowledge
- Voyager aims to address this, developing an LLM agent which can
  - **Propose suitable tasks** for itself
  - **Refine skills and commit them to memory** for future reuse
  - **Continually explore the world** and seek out new tasks

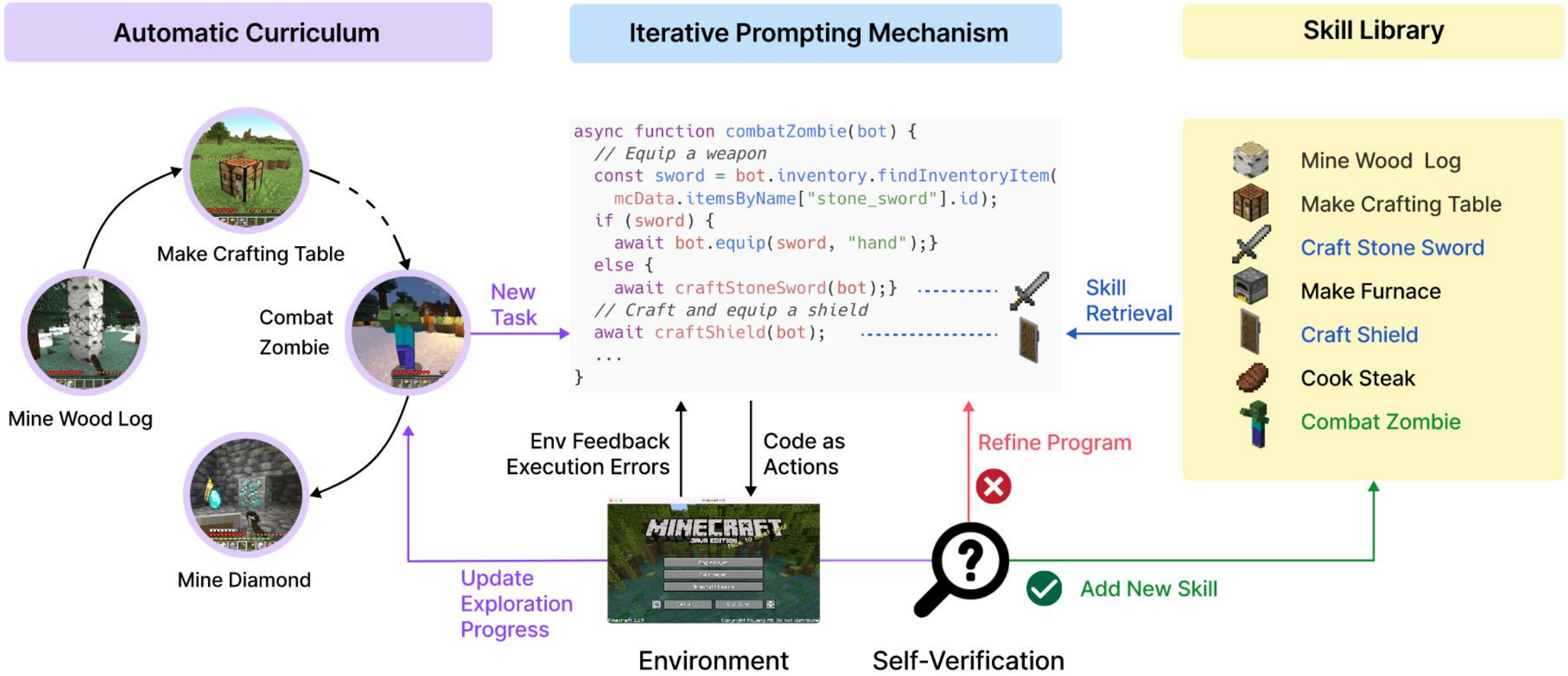


Figure 2: VOYAGER consists of three key components: an automatic curriculum for open-ended exploration, a skill library for increasingly complex behaviors, and an iterative prompting mechanism that uses code as action space.

Part 1: Automatic Curriculum

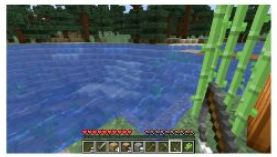
- How? Prompting!
- Input prompt consists of:
  a. **Directives encouraging diverse behaviors and imposing constraints**
  b. **The agent's current state** (inventory, equipment, nearby blocks...)
  c. **Previously completed and failed tasks** (reflecting its exploration progress and capabilities frontier)
  d. **Additional context:** Authors use GPT-3.5 to self ask and answer questions based on the current state and exploration progress

Part 1: Automatic Curriculum (cont.)

Inventory (5/36): {'oak_planks': 3, 'stick': 4, 'crafting_table': 1, 'stone': 3, 'wooden_pickaxe': 1}

GPT-4

Reasoning: Since you have a wooden pickaxe and some stones, it would be beneficial to upgrade your pickaxe to a stone pickaxe for better efficiency.
Task: Craft 1 stone pickaxe.



Biome: river
Inventory (4/36): {'sugar_cane': 3, 'paper': 3, 'fishing_rod': 1, 'wooden_pickaxe': 1}

GPT-4

Reasoning: Since you have a fishing rod in your inventory and you are near a river biome, it would be a good opportunity to catch some fish for food and experience.
Task: Catch 1 fish.

Nearby entities: pig, cat, villager
Health: 12/20
Hunger: 0/20

GPT-4

Reasoning: Your hunger is at 0, which means you need to find food to replenish your hunger. Since there are pigs nearby, you can kill one to obtain raw porkchops.
Task: Kill 1 pig.

Inventory (6/36): {'furnace': 1, 'stone_pickaxe': 1, 'oak_planks': 7, 'crafting_table': 1, 'raw_iron': 4, 'coal': 1}

GPT-4

Reasoning: You have raw iron and coal, and you have a furnace. It's time to smelt the iron to make iron ingots, which can be used to craft better tools and armor.
Task: Smelt 4 raw iron.

Time: night
Nearby entities: zombie
Equipment: 'stone_sword', 'shield'

GPT-4

Reasoning: Since it's night and there's a zombie nearby, it's a good opportunity to try killing the zombie now that you have a stone sword and a shield equipped.
Task: Kill 1 zombie.

Part 2: Skill Generation

- Input prompt consists of:
  a. **Guidelines for code generation,** encouraging generic and reusable code
  b. **Control primitive APIs, and relevant skills** retrieved from the skill library
  c. **The generated code from the last round, environment feedback, execution errors, and critique,** based on which GPT-4 can self-improve
  d. **The agent's current state,** including inventory, equipment, nearby blocks and entities, biome, time, health and hunger bars, and position
  e. **Chain-of-thought prompting** to do reasoning before code generation.

Part 2 (cont.): Skill Retrieval

Program Generated by GPT-4

```
async function combatZombie(bot) {
  // Equip a weapon
  const sword =
bot.inventory.findInventoryItem(
    mcData.itemsByName[
      "stone_sword"
    ].id
  );
  if (sword) {
    await bot.equip(sword, "hand");
  } else {
    await craftStoneSword(bot);
    ...
  }
  // Craft and equip a shield
  ...
  // Recover hunger
  ...
  // Look for and combat a zombie
  ...
}
```

GPT-3.5

Program Description

```
async function combatZombie(bot) {
  // The function is about
equipping a stone sword to combat
a zombie. If a stone sword is not
found, it will craft one.
Additionally, it crafts and equips
a shield for added protection.
Afterwards, it proceeds to cook
sticks in order to restore hunger.
Once hunger is replenished, it
actively searches for a zombie and
engages in combat with it.
}
```

Embedding → Key

Value

Add

Skill Library

Task: Craft Iron Pickaxe

How to craft an iron pickaxe in Minecraft?

GPT-3.5

To craft an iron pickaxe, you need to 3 iron ingots and 2 sticks. Once you have gathered the materials, ...

Environment Feedback

Embedding → Query → Skill Library

Retrieve

Top-5 Relevant Skills

Part 3: Iterative Prompting Mechanism

Environment Feedback

I cannot make stick because I need: 2 more planks
I cannot make stone_shovel because I need: 2 more stick

GPT-4

```
async function craftStoneShovelWithTable(bot) {
   // If not enough cobblestone, mine cobblestone
   ...
+  // If not enough sticks, check if there are
+  // enough planks in the inventory
   if (sticksCount < 2) {
+    const planksCount = bot.inventory.count(
+      mcData.itemsByName.acacia_planks.id);
+    if (planksCount < 2) {
+      // Collect or craft planks
+      await mineBlock(bot, "acacia_log", 1);
+      await craftItem(bot, "acacia_planks", 1);
+    }
+    // Craft sticks using planks
     await craftItem(bot, "stick", 1);}
   await craftItem(bot, "stone_shovel", 1);
}
```

Execution Error

throw new Error(`No item named ${name}`);
No item named acacia_axe
at line 18:await craftItem(bot, "acacia_axe", 1);

GPT-4

```
-async function craftAcaciaAxe(bot) {
+async function craftWoodenAxe(bot) {
   // Craft anacia planks and sticks
   ...

   // Place the crafting table near the bot
   ...

-  // Craft an acacia axe using 3 acacia planks
-  // and 2 sticks
-  await craftItem(bot, "acacia_axe", 1);
-  bot.chat("Acacia axe crafted.");
+  // Craft a wooden axe using 3 acacia planks
+  // and 2 sticks
+  await craftItem(bot, "wooden_axe", 1);
+  bot.chat("Wooden axe crafted.");
}
```

Part 3: Iterative Prompting Mechanism

Iterative process repeats **until self-verification validates the tasks completion**

Results

Table 1: Tech tree mastery. Fractions indicate the number of successful trials out of three total runs. 0/3 means the method fails to unlock a level of the tech tree within the maximal prompting iterations (160). Numbers are prompting iterations averaged over three trials. The fewer the iterations, the more efficient the method.

Method	Wooden Tool	Stone Tool	Iron Tool	Diamond Tool
ReAct [29]	N/A (0/3)	N/A (0/3)	N/A (0/3)	N/A (0/3)
Reflexion [30]	N/A (0/3)	N/A (0/3)	N/A (0/3)	N/A (0/3)
AutoGPT [28]	92 ± 72 (3/3)	94 ± 72 (3/3)	135 ± 103 (3/3)	N/A (0/3)
VOYAGER w/o Skill Library	$\mathbf{7 \pm 2}$ (3/3)	$\mathbf{9 \pm 4}$ (3/3)	29 ± 11 (3/3)	N/A (0/3)
VOYAGER (Ours)	$\mathbf{6 \pm 2}$ (3/3)	$\mathbf{11 \pm 2}$ (3/3)	$\mathbf{21 \pm 7}$ (3/3)	**102** (1/3)

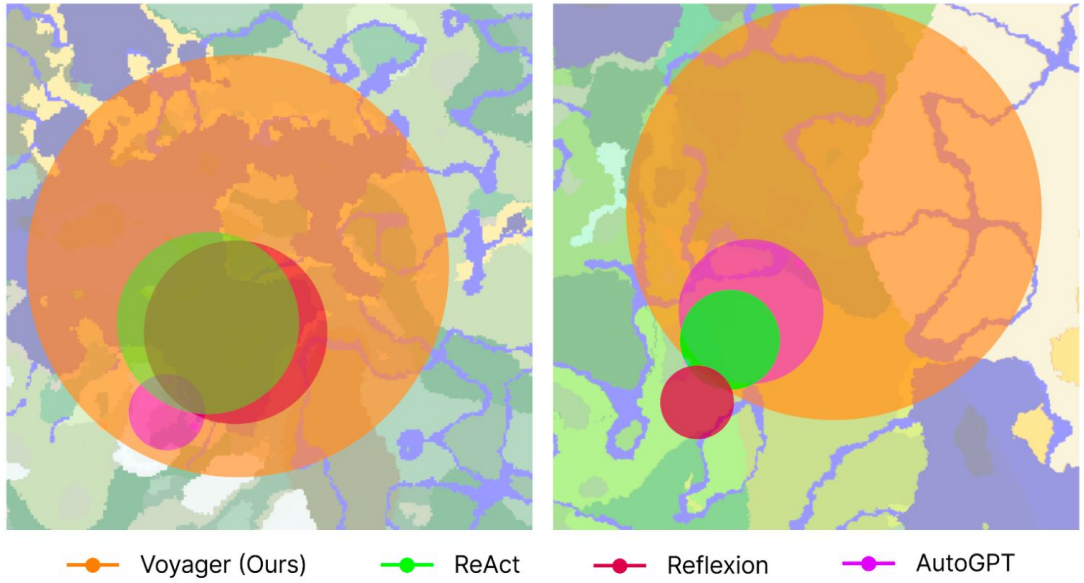


Figure 7: Map coverage: bird's eye views of Minecraft maps. VOYAGER is able to traverse 2.3× longer distances compared to baselines while crossing diverse terrains.

Results (Zero-Shot Generalization)

Method	Diamond Pickaxe	Golden Sword	Lava Bucket	Compass
ReAct [29]	N/A (0/3)	N/A (0/3)	N/A (0/3)	N/A (0/3)
Reflexion [30]	N/A (0/3)	N/A (0/3)	N/A (0/3)	N/A (0/3)
AutoGPT [28]	N/A (0/3)	N/A (0/3)	N/A (0/3)	N/A (0/3)
AutoGPT [28] w/ Our Skill Library	39 (1/3)	30 (1/3)	N/A (0/3)	30 (2/3)
VOYAGER w/o Skill Library	36 (2/3)	30 ± 9 (3/3)	27 ± 9 (3/3)	26 ± 3 (3/3)
VOYAGER (Ours)	**19 ± 3** (3/3)	**18 ± 7** (3/3)	**21 ± 5** (3/3)	**18 ± 2** (3/3)

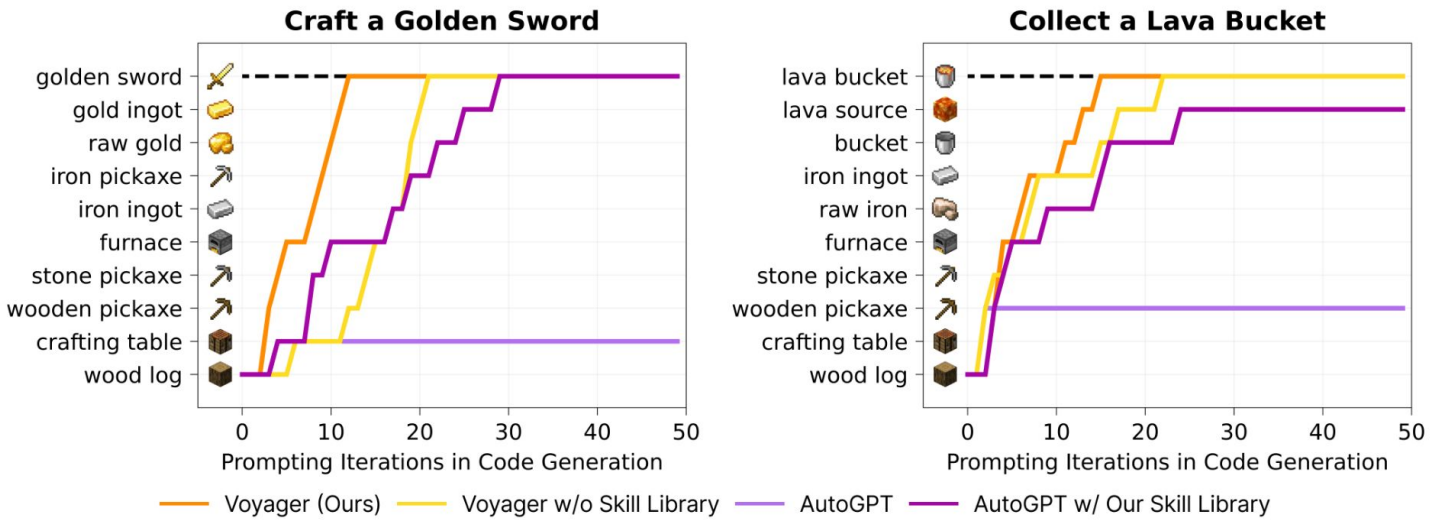


Figure 8: Zero-shot generalization to unseen tasks. We visualize the intermediate progress of each method on two tasks. See Appendix, Sec. B.4.3 for the other two tasks. We do not plot ReAct and Reflexion since they do not make any meaningful progress.

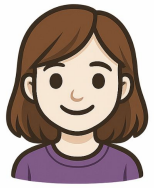
Key Insights and Enabling Ideas (tl;dr)

- By combining the **world knowledge** in pre-trained LLMs with the framework of program synthesis one is able to develop **highly-capable, lifelong learning agents**
- Importantly, all ‘learning’ is done through iterative prompting and skill caching/retrieval. No fine-tuning necessary!

ReGAL: Refactoring Programs to Discover Generalizable Abstractions

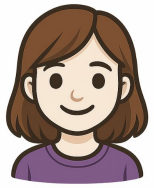
Stengel-Eskin et al.
ICML 2024

Motivation



Write me a function that generate array size 100 with random numbers

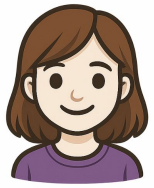
Motivation



Write me a function that generate array size 100 with random numbers

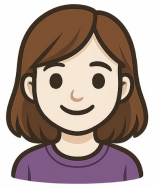
```
def generate_random_array():
    return [random.random() for _ in range(100)]
```

Motivation



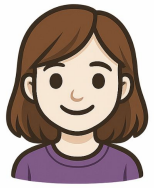
Write me a function that generate array size 100 with random numbers

```
def generate_random_array():
    return [random.random() for _ in range(100)]
```



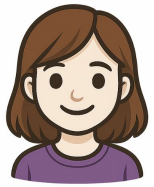
Write me a function that generate a matrix size 100*100 with random numbers

Motivation



Write me a function that generate array size 100 with random numbers

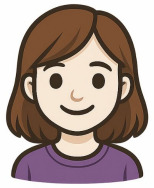
```
def generate_random_array():
    return [random.random() for _ in range(100)]
```



Write me a function that generate a matrix size 100*100 with random numbers

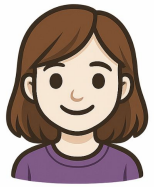
What did Dori wish?

Motivation



Write me a function that generate array size 100 with random numbers

```
def generate_random_array():
    return [random.random() for _ in range(100)]
```

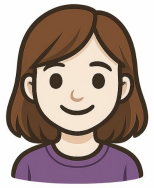


Write me a function that generate a matrix size 100*100 with random numbers

What did Dori wish?

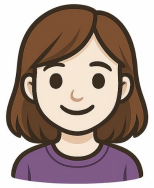
```
def generate_random_matrix():
    return [generate_random_array() for _ in range(100)]
```

Motivation



Write me a function that generate array size 100 with random numbers

```
def generate_random_array():
    return [random.random() for _ in range(100)]
```



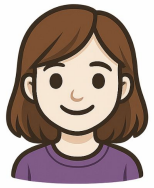
Write me a function that generate a matrix size 100*100 with random numbers

What did Dori wish?

```
def generate_random_matrix():
    return [generate_random_array() for _ in range(100)]
```

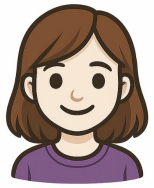
What was the output?

Motivation



Write me a function that generate array size 100 with random numbers

```
def generate_random_array():
    return [random.random() for _ in range(100)]
```



Write me a function that generate a matrix size 100*100 with random numbers

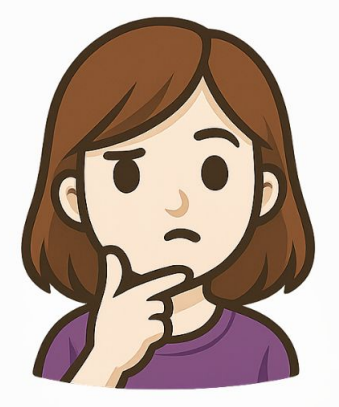
What did Dori wish?

```
def generate_random_matrix():
    return [generate_random_array() for _ in range(100)]
```

What was the output?

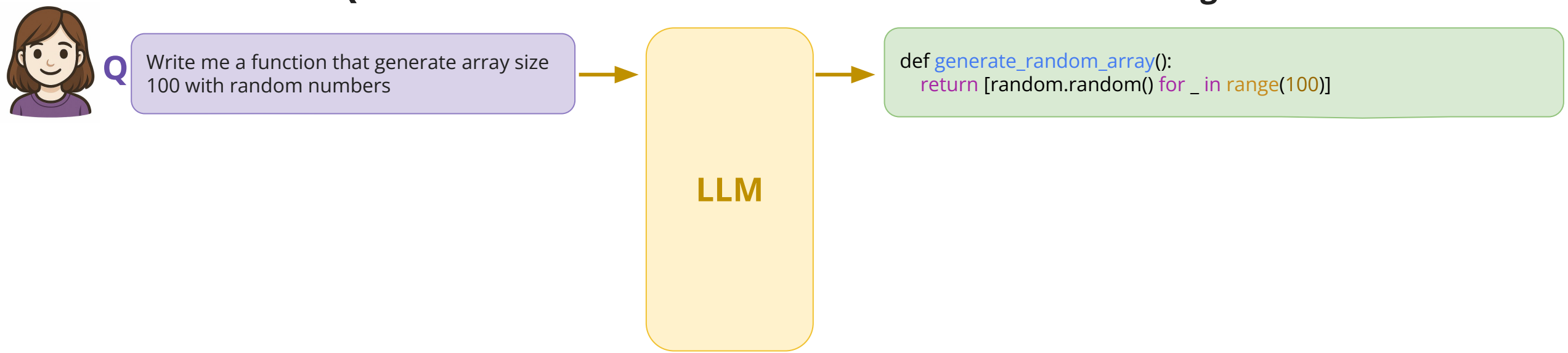
```
def generate_random_matrix():
    return [[random.random() for _ in range(100)] for _ in range(100)]
```

Motivation

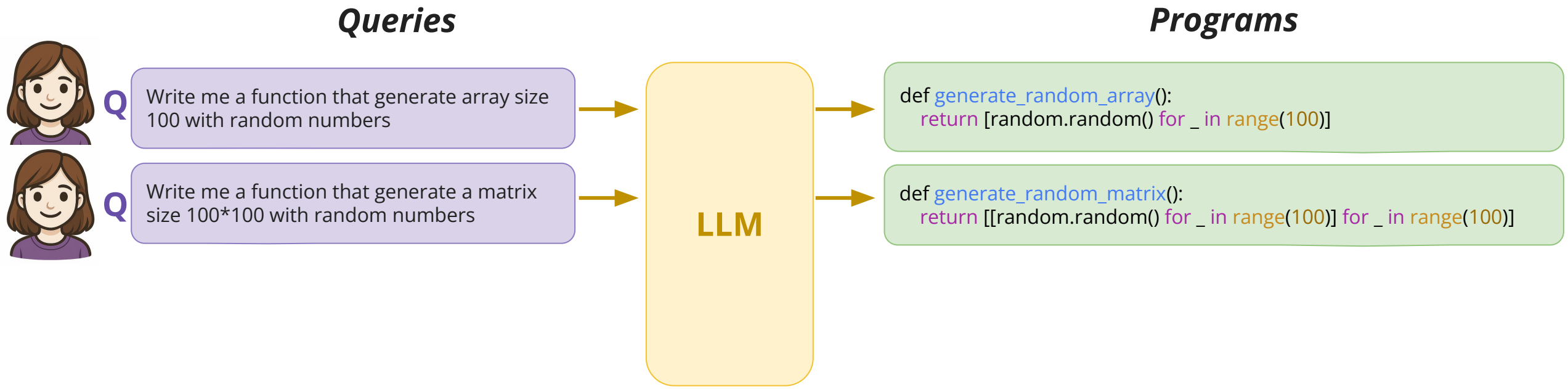


What exactly happened?

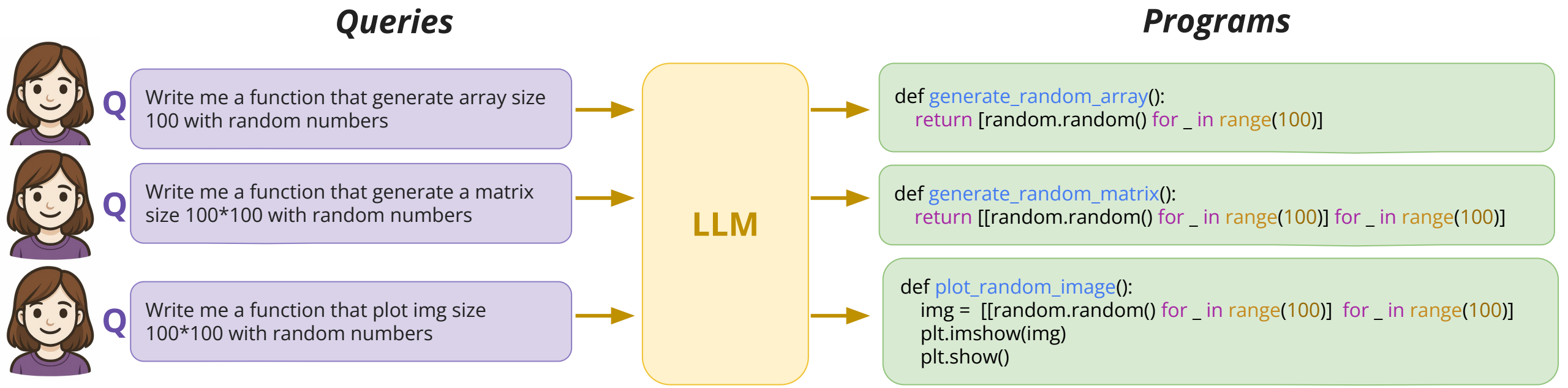
Motivation


Queries
Programs
Q
Write me a function that generate array size 100 with random numbers
LLM
def generate_random_array():
return [random.random() for _ in range(100)]

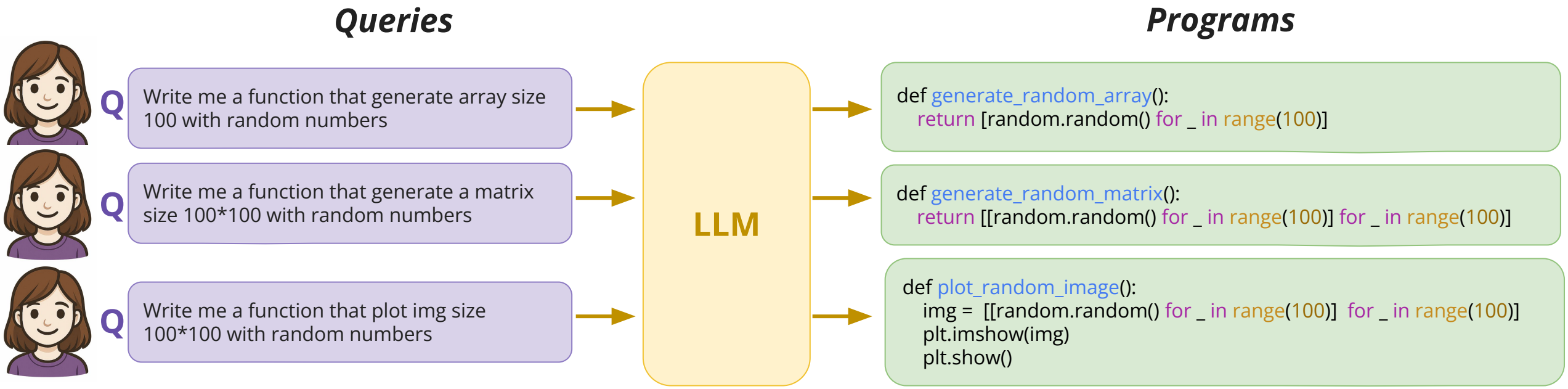
Motivation


Queries
Programs
Q
Write me a function that generate array size 100 with random numbers
Q
Write me a function that generate a matrix size 100*100 with random numbers
LLM
def generate_random_array():
return [random.random() for _ in range(100)]
def generate_random_matrix():
return [[random.random() for _ in range(100)] for _ in range(100)]

Motivation


Queries
Programs
Q
Write me a function that generate array size 100 with random numbers
Q
Write me a function that generate a matrix size 100*100 with random numbers
Q
Write me a function that plot img size 100*100 with random numbers
LLM
def generate_random_array():
return [random.random() for _ in range(100)]
def generate_random_matrix():
return [[random.random() for _ in range(100)] for _ in range(100)]
def plot_random_image():
img = [[random.random() for _ in range(100)] for _ in range(100)]
plt.imshow(img)
plt.show()

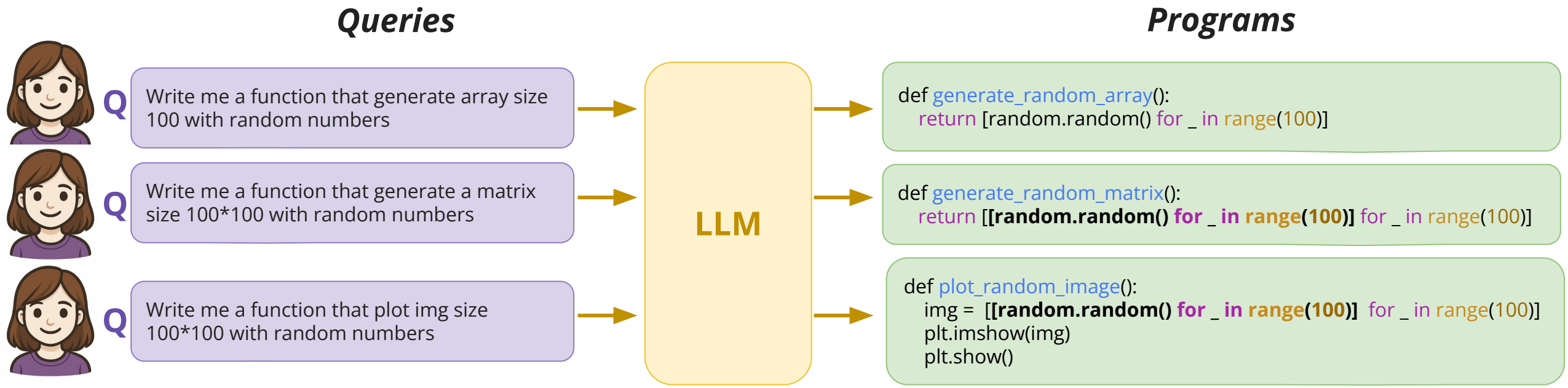
Motivation



Each program is generated independently with no reference to other queries or programs

The Problem

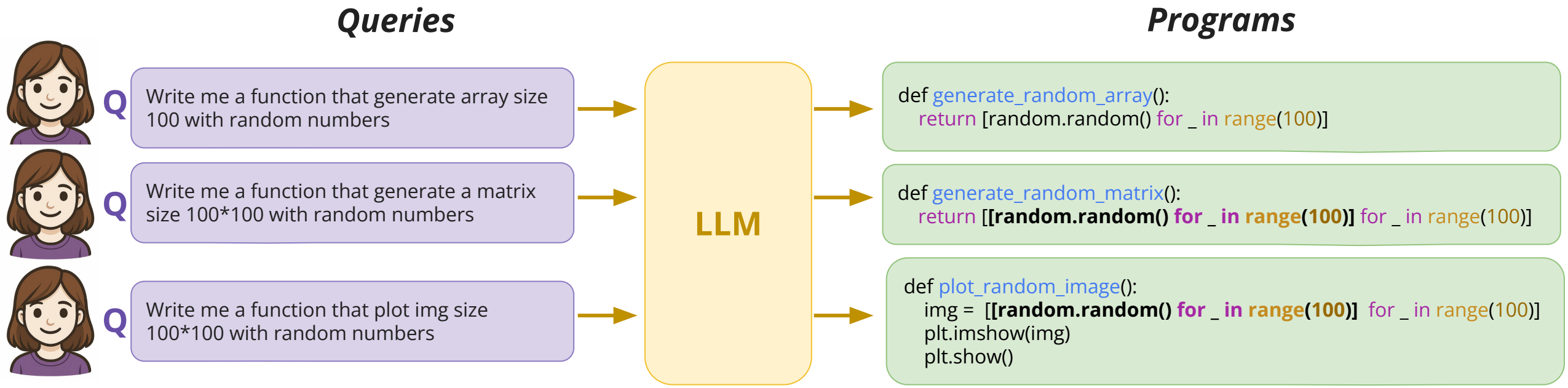
Lack of Reusability



We could have used generate_random_array to create other programs

The Problem

Lack of Abstraction



Shared abstractions can improve accuracy by making skills more accessible to the model.

Related Work

DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning

Kevin Ellis,[1,4,5] Catherine Wong,[1,4,5] Maxwell Nye,[1,4,5]
Mathias Sablé-Meyer,[1,3] Luc Cary,[1] Lucas Morales,[1,4,6] Luke Hewitt,[1,4,5]
Armando Solar-Lezama,[1,2,6] Joshua B. Tenenbaum[1,2,4,5]

[1]MIT [2]CSAIL [3]NeuroSpin [4]Center for Brains, Minds, and Machines

[5]Department of Brain and Cognitive Sciences [6]Department of Electrical Engineering and Computer Science

Related Work

DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning

Assume domain language is λ-calculus

Armando Solar-Lezama,[1,2,6] Joshua B. Tenenbaum[1,2,4,5]

[1]MIT [2]CSAIL [3]NeuroSpin [4]Center for Brains, Minds, and Machines

[5]Department of Brain and Cognitive Sciences [6]Department of Electrical Engineering and Computer Science

Related Work

VOYAGER: An Open-Ended Embodied Agent with Large Language Models

Guanzhi Wang[1 2] ✉, **Yuqi Xie**[3], **Yunfan Jiang**[4*], **Ajay Mandlekar**[1*],
Chaowei Xiao[1 5], **Yuke Zhu**[1 3], **Linxi "Jim" Fan**[1†] ✉, **Anima Anandkumar**[1 2†]
[1]NVIDIA, [2]Caltech, [3]UT Austin, [4]Stanford, [5]UW Madison
*Equal contribution †Equal advising ✉ Corresponding authors
https://voyager.minedojo.org

Related Work

VOYAGER: An Open-Ended Embodied Agent with Large Language Models

Does not involve refactoring of the code, only skill learning

Guanzhi Wang[1,2✉], Yuqi Xie[3], Yunfan Jiang[4*], Ajay Mandlekar[1*],
Chaowei Xiao[1,5], Yuke Zhu[1,3], Linxi "Jim" Fan[1†✉], Anima Anandkumar[1,2†]
[1]NVIDIA, [2]Caltech, [3]UT Austin, [4]Stanford, [5]UW Madison
*Equal contribution †Equal advising ✉ Corresponding authors
https://voyager.minedojo.org

Solution

ReGAL: Refactoring for **G**eneralizable **A**bstraction **L**earning

Queries

Q Write me a function that generate array size 100 with random numbers

Q Write me a function that generate a matrix size 100*100 with random numbers

Q Write me a function that plot img size 100*100 with random numbers

Programs

P

```
def generate_random_array():
    return [random.random() for _ in range(100)]
```

P

```
def generate_random_matrix():
    return [[random.random() for _ in range(100)] for _ in range(100)]
```

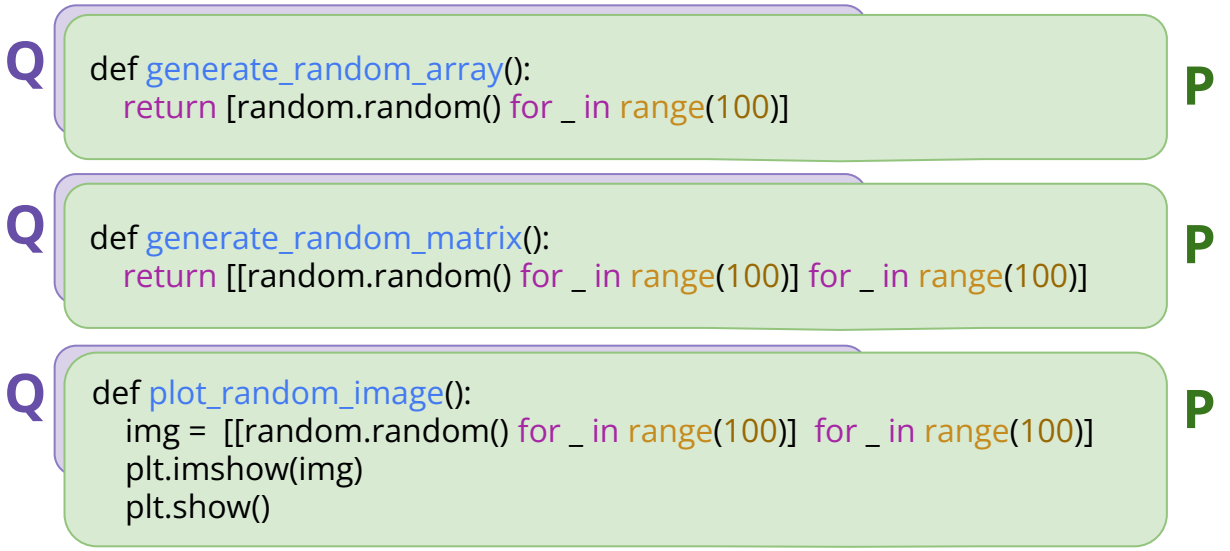
P

```
def plot_random_image():
    img =  [[random.random() for _ in range(100)]  for _ in range(100)]
    plt.imshow(img)
    plt.show()
```

Solution

ReGAL: Refactoring for **G**eneralizable **A**bstraction **L**earning

Programs + Queries



Solution

ReGAL: Refactoring for **G**eneralizable **A**bstraction **L**earning

Code Bank (C)

```
def _rand_list(n, rnd=random.random):
  return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
  return [_rand_list(c, rnd) for _ in range(r)]

def _show_image(img):
  plt.imshow(matrix)
  plt.show()
```

Helper functions

Demo Bank (D)

Q
```
def generate_random_array():
  return _rand_list(100)
```
P

Q
```
def generate_random_matrix():
  return _rand_matrix(100, 100)
```
P

Q
```
def generate_random_image():
  _show_image(_rand_matrix(100, 100))
```
P

Examples

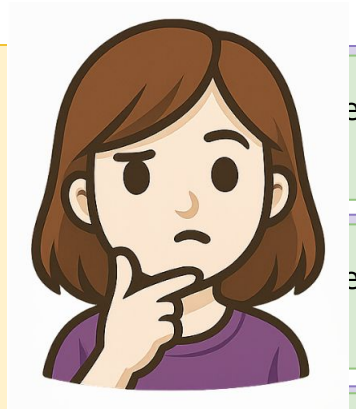
Solution

ReGAL: Refactoring for **G**eneralizable **A**bstraction **L**earning

Code Bank (C)

```
def _rand_list(n, rnd=random.random):
  return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
  return [_rand_list(c, rnd) for _ in range(r)]

def _show_image(img):
  plt.imshow(matrix)
  plt.show()
```

Helper functions



Q

Demo Bank (D)

```
ef generate_random_array():
 return _rand_list(100)
```

P

```
ef generate_random_matrix():
 return _rand_matrix(100, 100)
```

P

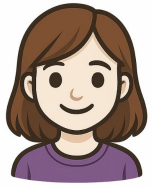
```
def generate_random_image():
    _show_image(_rand_matrix(100, 100))
```

P

Examples

ReGAL Training

Stage 1: Refactoring Examples



Q Refactor this program with reusable helper functions

Q

```
def generate_random_array():
    return [random.random() for _ in range(100)]
```

P

Q

```
def generate_random_matrix():
    return [[random.random() for _ in range(100)] for _ in range(100)]
```

P

Q

```
def plot_random_image():
    img =  [[random.random() for _ in range(100)]  for _ in range(100)]
    plt.imshow(img)
    plt.show()
```

P

ReGAL Training

Programs

```
def generate_random_array():
    return [random.random() for _ in range(100)]
```

```
def generate_random_matrix():
    return [[random.random() for _ in range(100)] for _ in range(100)]
```

```
def plot_random_image():
    img =  [[random.random() for _ in range(100)]  for _ in range(100)]
    plt.imshow(img)
    plt.show()
```

Refactor *Programs*

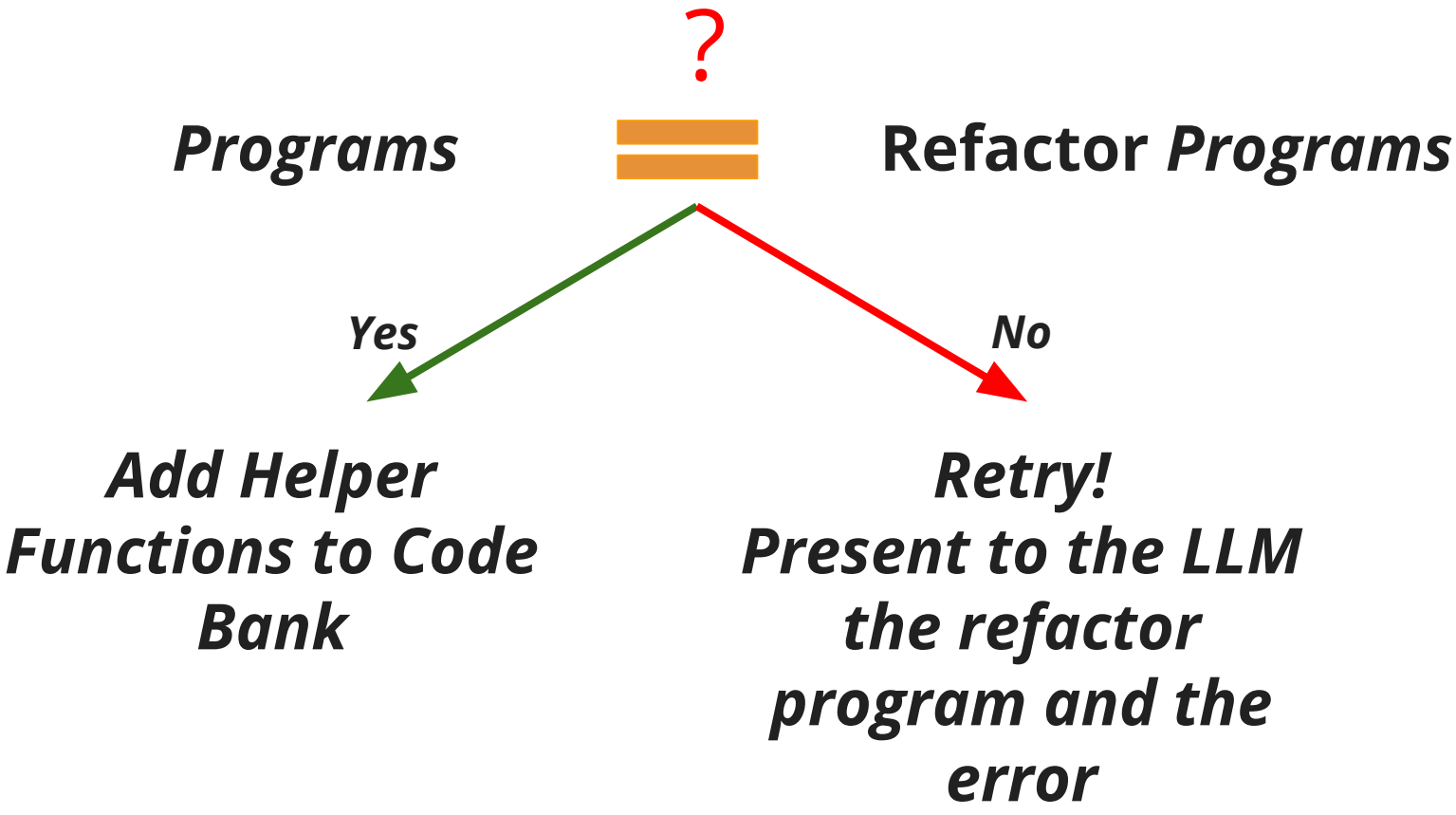
```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def generate_random_array():
    return _rand_list(100)
```

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
    return [_rand_list(c, rnd) for _ in range(r)]

def generate_random_matrix():
    return _rand_matrix(100, 100)
```

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
    return [_rand_list(c, rnd) for _ in range(r)]

def _show_image(img):
    plt.imshow(matrix)
    plt.show()

def generate_random_image():
    _show_image(_rand_matrix(100, 100))
```

ReGAL Training

Stage 2: Verification and Feedback-based Retrial



ReGAL Training

Programs

```
def generate_random_array():
    return [random.random() for _ in range(100)]
```

?

Refactor *Programs*

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def generate_random_array():
    return _rand_list(100)
```

ReGAL Training

Programs

```
def generate_random_array():
    return [random.random() for _ in range(100)]
```

Refactor *Programs*

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def generate_random_array():
    return _rand_list(100)
```

ReGAL Training

Programs

```
def generate_random_array():
    return [random.random() for _ in range(100)]
```

Refactor *Programs*

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def generate_random_array():
    return _rand_list(100)
```

Code Bank (C)

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]
```

ReGAL Training

Programs

```
def generate_random_array():
    return [random.random() for _ in range(100)]
```

Refactor *Programs*

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def generate_random_array():
    return _rand_list(100)
```

Code Bank (C)

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]
```

Demo Bank (D)

Q

```
def generate_random_array():
    return _rand_list(100)
```

P

ReGAL Training

Programs

```
def generate_random_matrix():
    return [[random.random() for _ in range(100)] for _ in range(100)]
```

?

Refactor *Programs*

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
    return [_rand_list(c, rnd) for _ in range(r)]

def generate_random_matrix():
    return _rand_matrix(100, 100)
```

Code Bank (C)

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]
```

Demo Bank (D)

Q

```
def generate_random_array():
    return _rand_list(100)
```

P

ReGAL Training

Programs

```
def generate_random_matrix():
    return [[random.random() for _ in range(100)] for _ in range(100)]
```

Refactor *Programs*

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
    return [_rand_list(c, rnd) for _ in range(r)]

def generate_random_matrix():
    return _rand_matrix(100, 100)
```

Code Bank (C)

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]
```

Demo Bank (D)

Q

```
def generate_random_array():
    return _rand_list(100)
```

P

ReGAL Training

Programs

```
def generate_random_matrix():
    return [[random.random() for _ in range(100)] for _ in range(100)]
```

Refactor *Programs*

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
    return [_rand_list(c, rnd) for _ in range(r)]

def generate_random_matrix():
    return _rand_matrix(100, 100)
```

Code Bank (C)

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
    return [_rand_list(c, rnd) for _ in range(r)]
```

Demo Bank (D)

Q

```
def generate_random_array():
    return _rand_list(100)
```

P

ReGAL Training

Programs

```
def generate_random_matrix():
    return [[random.random() for _ in range(100)] for _ in range(100)]
```

Refactor *Programs*

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
    return [_rand_list(c, rnd) for _ in range(r)]

def generate_random_matrix():
     return _rand_matrix(100, 100)
```

Code Bank (C)

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
    return [_rand_list(c, rnd) for _ in range(r)]
```

Demo Bank (D)

Q

```
def generate_random_array():
    return _rand_list(100)
```

P

Q

```
def generate_random_matrix():
    return _rand_matrix(100, 100)
```

P

ReGAL Training

Code Bank (C)

```
def _rand_list(n, rnd=random.random):
    return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
    return [_rand_list(c, rnd) for _ in range(r)]

def _show_image(img):
    plt.imshow(matrix)
    plt.show()
```

Demo Bank (D)

Q

```
def generate_random_array():
    return _rand_list(100)
```

P

Q

```
def generate_random_matrix():
    return _rand_matrix(100, 100)
```

P

Q

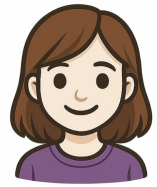
```
def generate_random_image():
    _show_image(_rand_matrix(100, 100))
```

P

ReGAL Training

Stage 3: Editing and Pruning

Code Bank Editing



Q

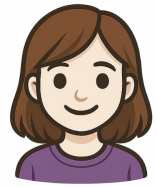
Function _show_image pass 3 of 6 tests, improve it

```
def _show_image(img):
    plt.imshow(matrix)
    plt.show()
```

ReGAL Training

Stage 3: Editing and Pruning

Code Bank Editing



Q

Function _show_image pass 3 of 6 tests, improve it

```
def _show_image(img, cmap=None):
  plt.imshow(matrix, cmap=cmap)
  plt.show()
```

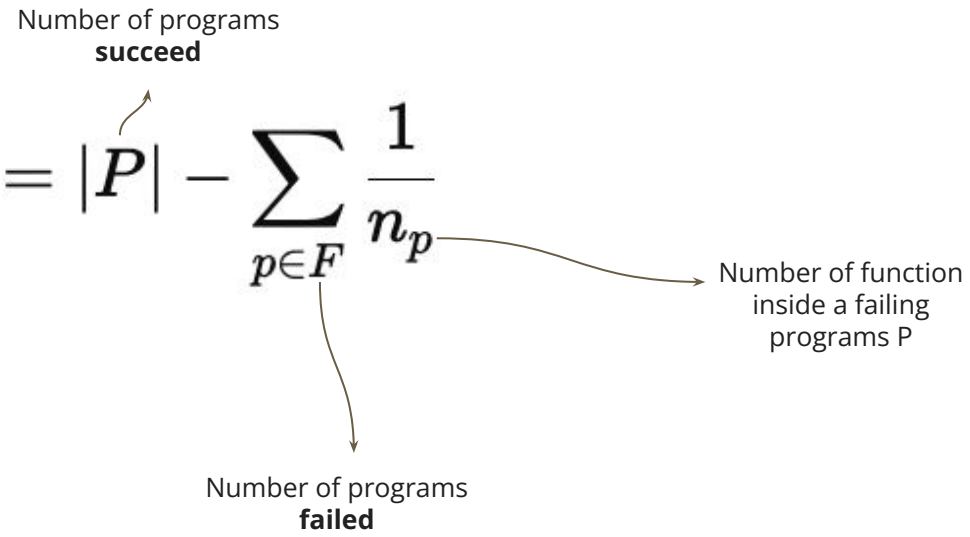
if more tests pass
update the code bank and the demo bank

ReGAL Training

Stage 3: Editing and Pruning

Code Bank Pruning

Code Bank Function Score



if s < 0 the function is **pruned** (removed).

ReGAL Training

Stage 3: Editing and Pruning

Code Bank Pruning

Code Bank Function Score

$$s = |P| - \sum_{p \in F} \frac{1}{n_p}$$

Number of programs **succeed**

Number of function inside a failing programs P

Number of programs **failed**

_show_image function used in 3 different programs

2 programs succeed, 1 failed.

The failed function contain 6 different help function calls

Overall _show_image_score: 2 - ⅙

if s < 0 the function is **pruned** (removed).

ReGAL Training

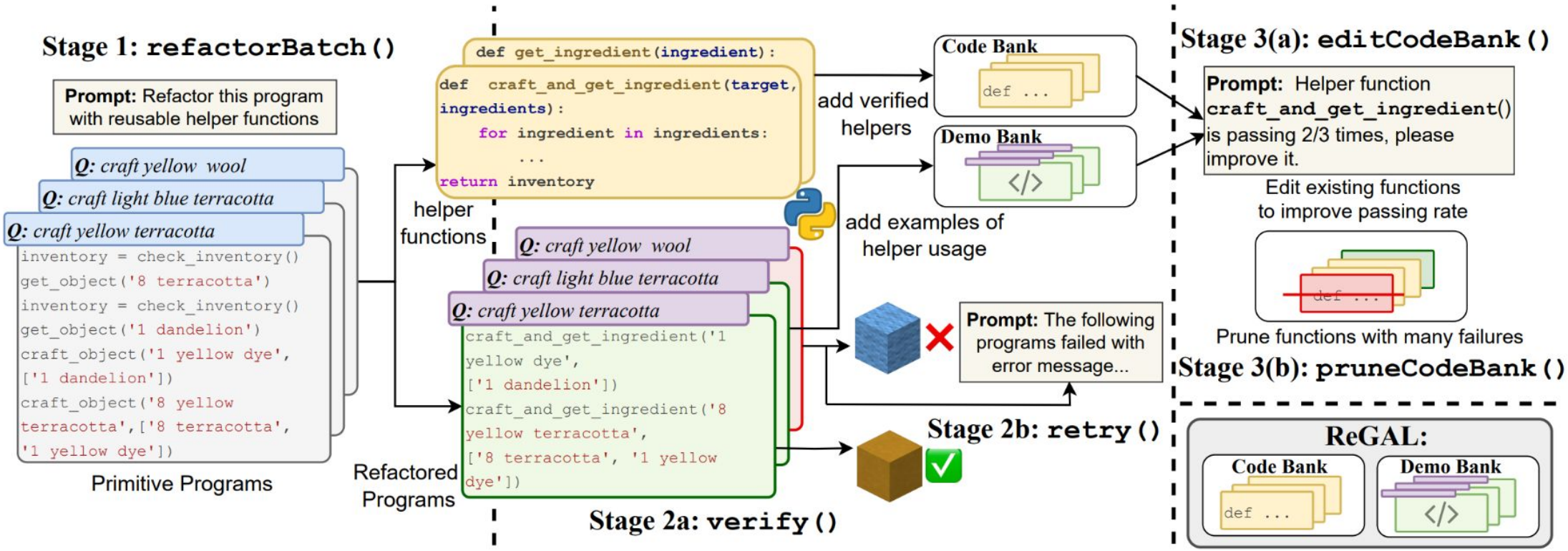


Figure 2. REGAL starts by refactoring a batch of primitive programs to develop a set of modified programs and helper functions (**Stage 1**). It then verifies the results of refactored programs, optionally retrying failed programs according to environment feedback. Useful helper functions are added to the Code Bank along with example usage added to the Demo Bank (**Stage 2**). Periodically, we edit and prune the Code Bank to improve its functions (**Stage 3**). At test time, the REGAL agent has access to the Code Bank, the Demo Bank, and the remaining original programs. It is compared against a baseline agent which has access to a larger number of original programs.

ReGAL Testing

Code Bank (C)

```
def _rand_list(n, rnd=random.random):
  return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
  return [_rand_list(c, rnd) for _ in range(r)]

def _show_image(img):
  plt.imshow(matrix)
  plt.show()
```

Demo Bank (D)

Q

```
def generate_random_array():
  return _rand_list(100)
```

P

Q

```
def generate_random_matrix():
  return _rand_matrix(100, 100)
```

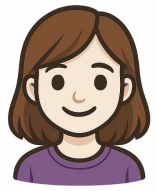
P

Q

```
def generate_random_image():
  _show_image(_rand_matrix(100, 100))
```

P

ReGAL Testing



Q Write me a function that visualize 10 random images in a loop

Code Bank (C)

```
def _rand_list(n, rnd=random.random):
  return [rnd() for  _ in range(n)]

def _rand_matrix(r ,c, rnd=random.random):
  return [_rand_list(c, rnd) for _ in range(r)]

def _show_image(img):
  plt.imshow(matrix)
  plt.show()
```

Demo Bank (D)

Q

```
def generate_random_array():
  return _rand_list(100)
```

P

Q

```
def generate_random_matrix():
  return _rand_matrix(100, 100)
```

P

Q

```
def generate_random_image():
  _show_image(_rand_matrix(100, 100))
```

P

```
def generate_ten_images():
  for _ in range(10):
    _show_image(_rand_matrix(100, 100))
```

Evaluation

Table 1. Accuracy of baseline agents predicting primitive programs (Prim.) and those augmented with REGAL helper functions (3 random seeds). Across domains and models, REGAL improves over a strong baseline agent with access to the same number of ICL examples. Math domains with no clear domain language marked with *.

	LOGO		Date		TextCraft		MATH (Alg.)*		TabMWP*	
Agent	**Prim.**	**REGAL**	**Prim.**	**REGAL**	**Prim.**	**REGAL**	**Prim.**	**REGAL**	**Prim.**	**REGAL**
CodeLlama-7B	$34.5_{\pm 1.3}$	$34.5_{\pm 1.6}$	$52.4_{\pm 0.7}$	$55.2_{\pm 1.4}$	$12.8_{\pm 1.3}$	$16.7_{\pm 1.3}$	$6.8_{\pm 0.8}$	$17.6_{\pm 0.8}$	$16.7_{\pm 1.6}$	$27.7_{\pm 0.6}$
CodeLlama-13B	$45.6_{\pm 0.3}$	$\mathbf{57.1}_{\pm 0.6}$	$42.8_{\pm 2.0}$	$68.9_{\pm 1.6}$	$18.8_{\pm 0.7}$	$26.9_{\pm 2.2}$	$14.0_{\pm 0.9}$	$17.6_{\pm 0.8}$	$29.1_{\pm 0.6}$	$27.9_{\pm 0.4}$
CodeLlama-34B	$50.2_{\pm 0.8}$	$50.8_{\pm 0.6}$	$47.2_{\pm 1.5}$	$68.5_{\pm 2.1}$	$22.2_{\pm 0.7}$	$\mathbf{30.8}_{\pm 1.3}$	$14.9_{\pm 1.1}$	$22.5_{\pm 1.6}$	$19.4_{\pm 1.2}$	$27.0_{\pm 0.8}$
Lemur-70B	$44.1_{\pm 1.4}$	$56.8_{\pm 0.9}$	$68.2_{\pm 0.4}$	$70.5_{\pm 0.6}$	$15.7_{\pm 1.7}$	$23.5_{\pm 2.1}$	$14.0_{\pm 1.2}$	$13.1_{\pm 1.2}$	$27.5_{\pm 0.9}$	$25.7_{\pm 0.8}$
GPT-3.5-turbo	$36.9_{\pm 1.6}$	$49.3_{\pm 1.1}$	$88.9_{\pm 0.3}$	$\mathbf{90.2}_{\pm 0.5}$	$15.4_{\pm 1.3}$	$18.4_{\pm 2.0}$	$43.2_{\pm 1.6}$	$\mathbf{55.4}_{\pm 2.8}$	$87.4_{\pm 0.9}$	$\mathbf{89.2}_{\pm 0.8}$

Evaluation

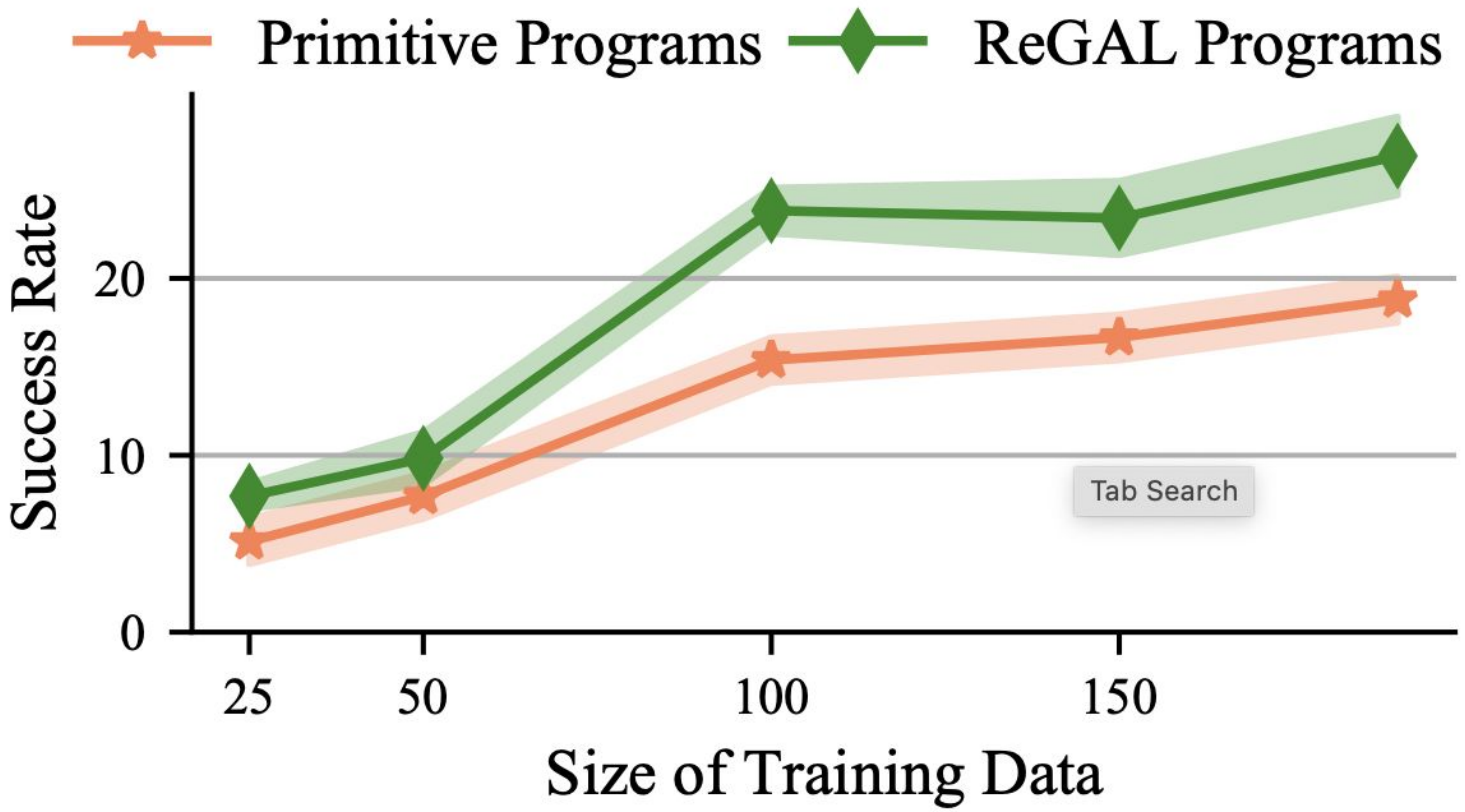


Figure 4. REGAL programs yield a higher success rate (accuracy) compared to primitive programs on TextCraft for different sizes of training set X using CodeLlama-13B.

Evaluation

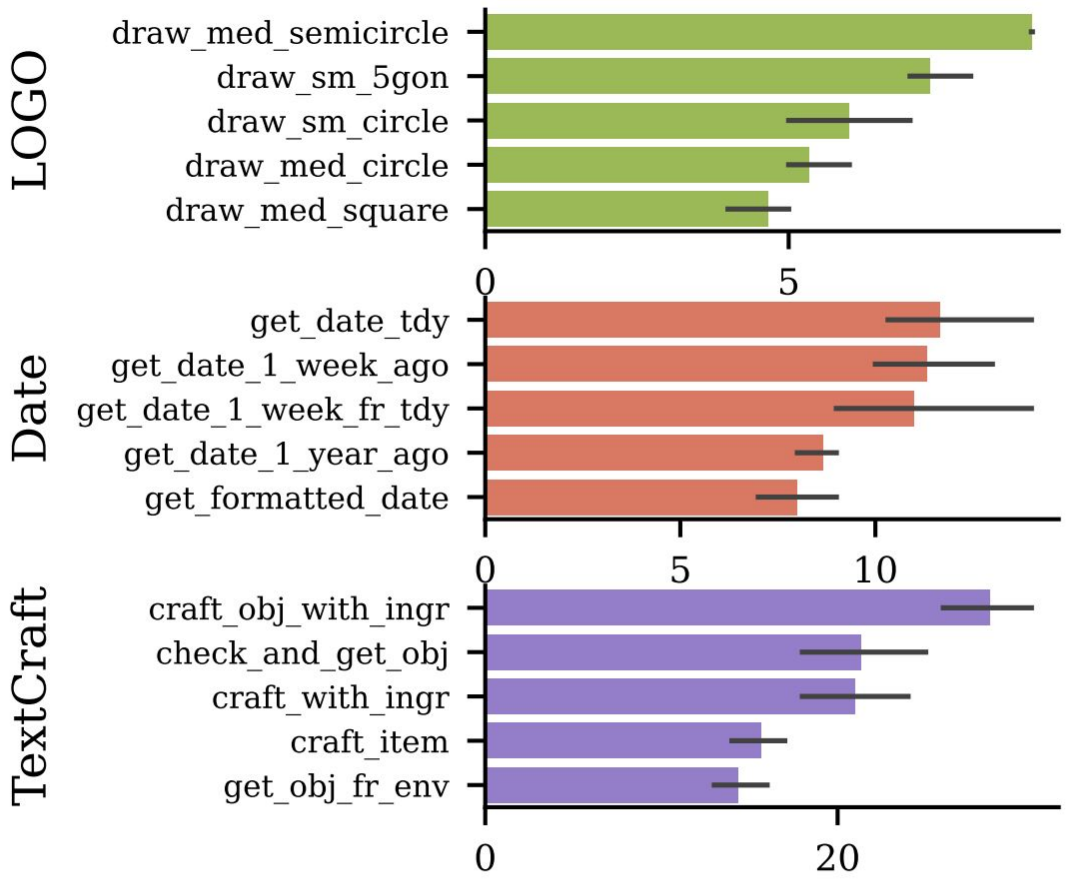
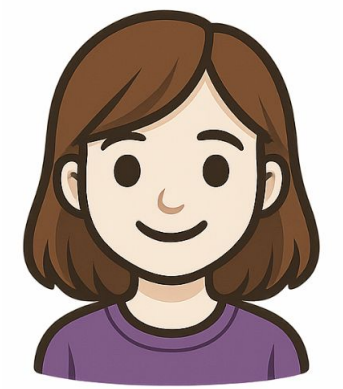
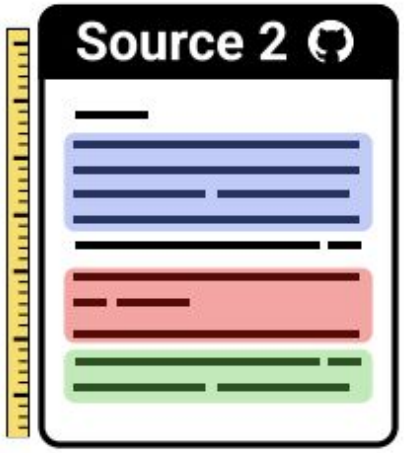


Figure 3. Function usage by CodeLlama-13B for the top-5 most common helpers illustrating reusability across examples. The x-axis denotes the number of times a functions is used in the test set.

Motivation

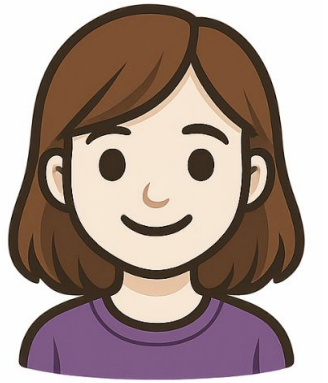
Dori had a very productive week

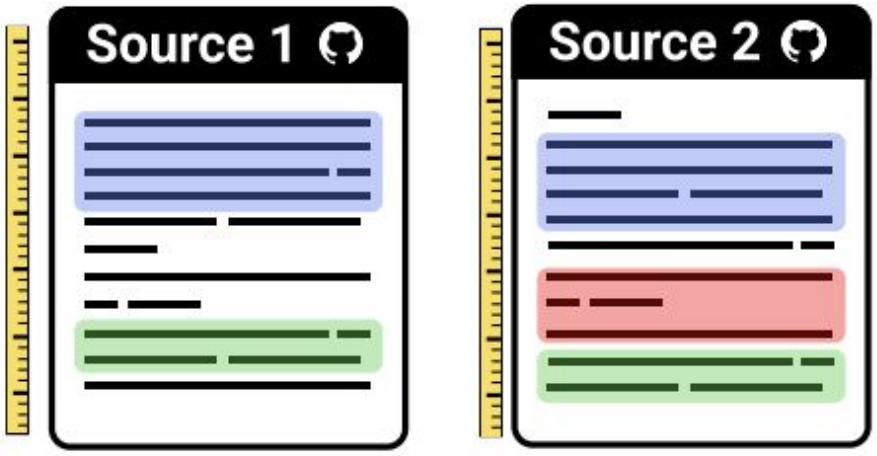




Motivation

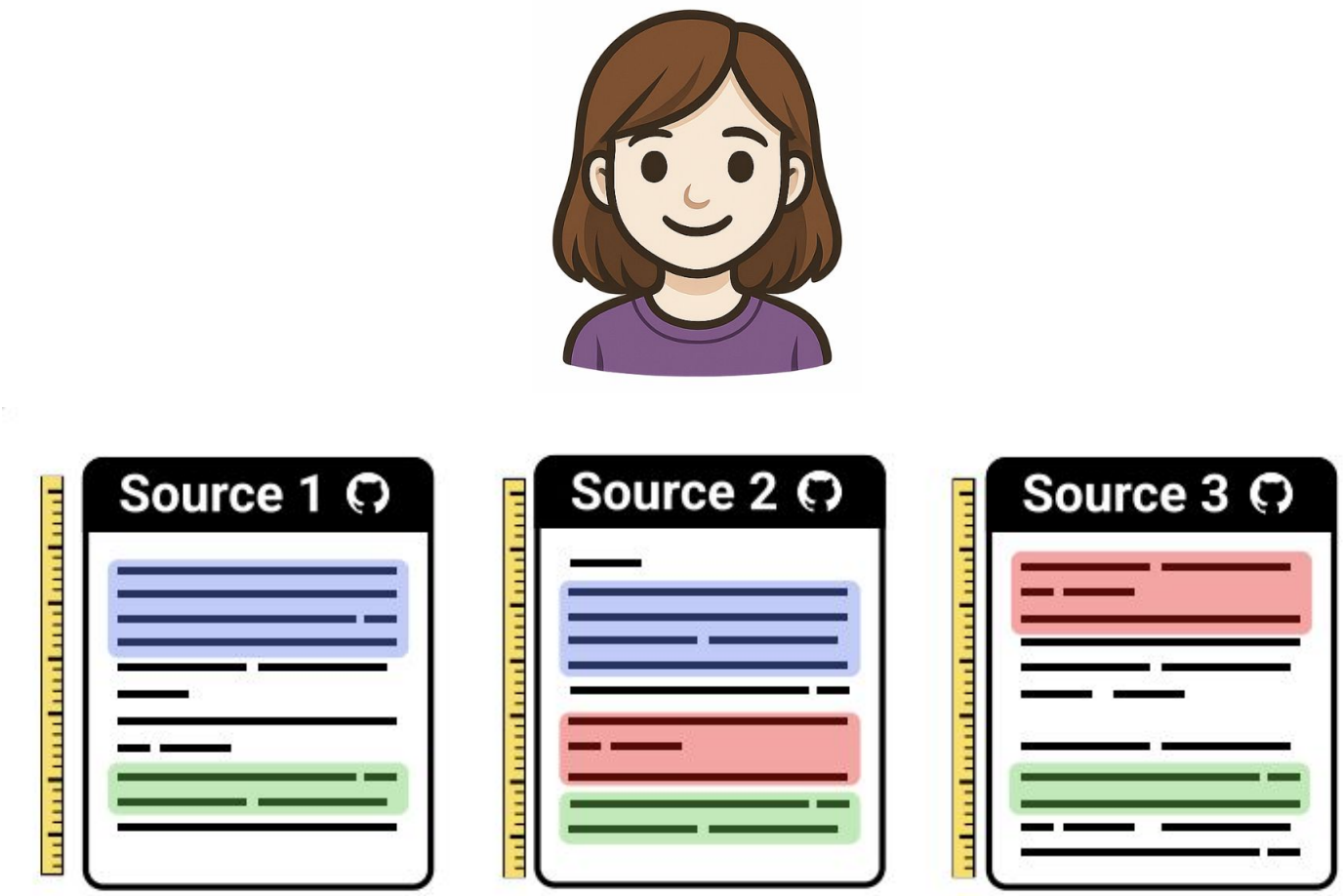
Dori had a very productive week





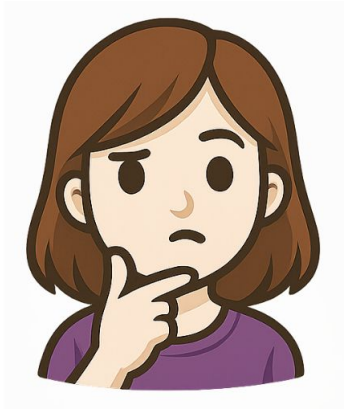
Motivation

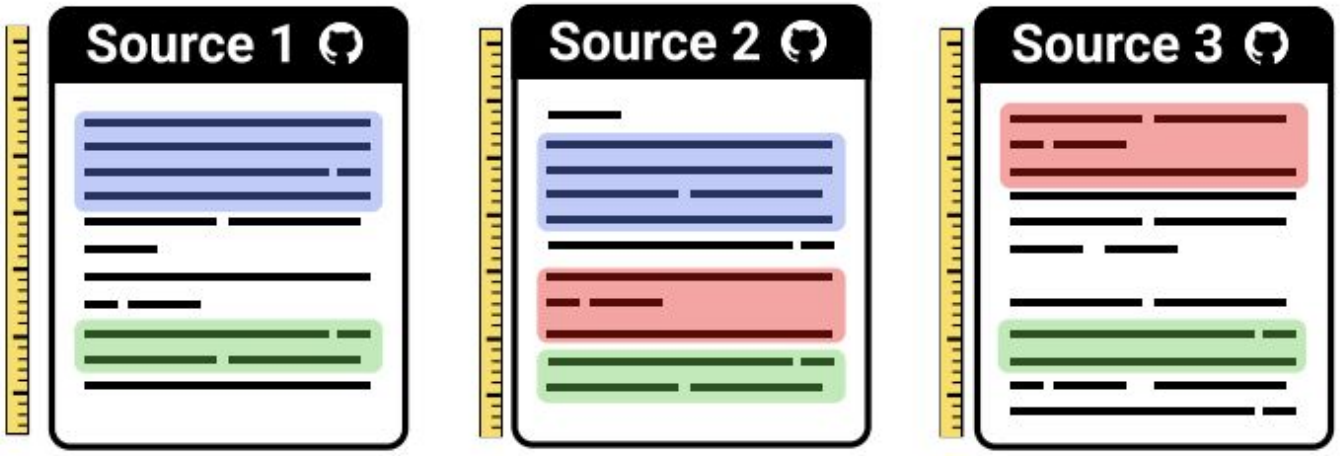
Dori had a very productive week



Motivation

Dori wants to extract common code for reuse





Refactoring Codebases through Library Design

Kovačič et al.
arXiv 2025

Motivation

- Create a cohesive library

Motivation

- Create a cohesive library
- That captures shared abstractions

Motivation

- Create a cohesive library
- That captures shared abstractions
- Must reduce the total code size

Motivation

- Create a cohesive library
- That captures shared abstractions
- Must reduce the total code size
- While supporting all original use cases

Motivation

- Create a cohesive library
- That captures shared abstractions
- Must reduce the total code size
- While supporting all original use cases
- And potentially opening up new use cases

Motivation

This is accomplished by searching for refactorings that are both correct and simple

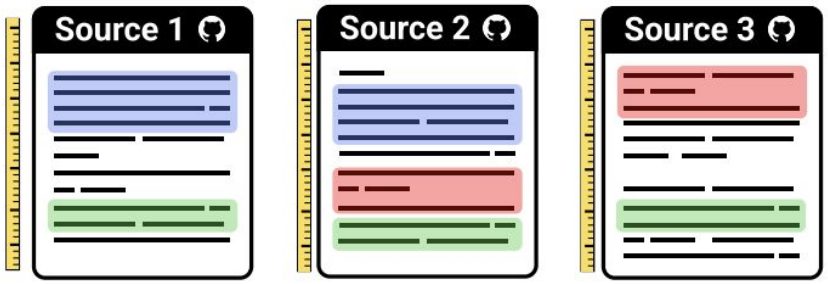
Motivation

This is accomplished by searching for refactorings that are both correct and simple

Motivation

Given a collection of code sources

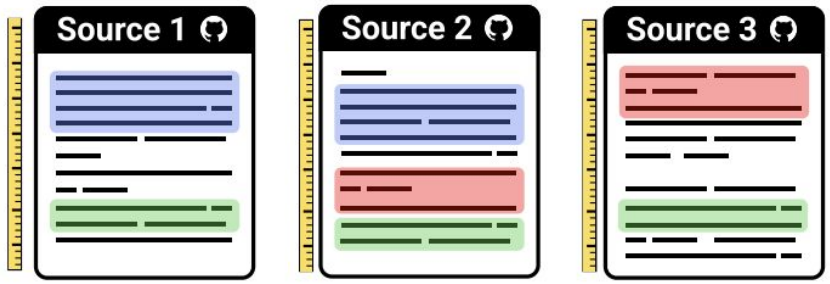
$$\{\rho_n\}_{n=1}^{N}$$



Motivation

Given a collection of code sources

$$\{\rho_n\}_{n=1}^{N}$$

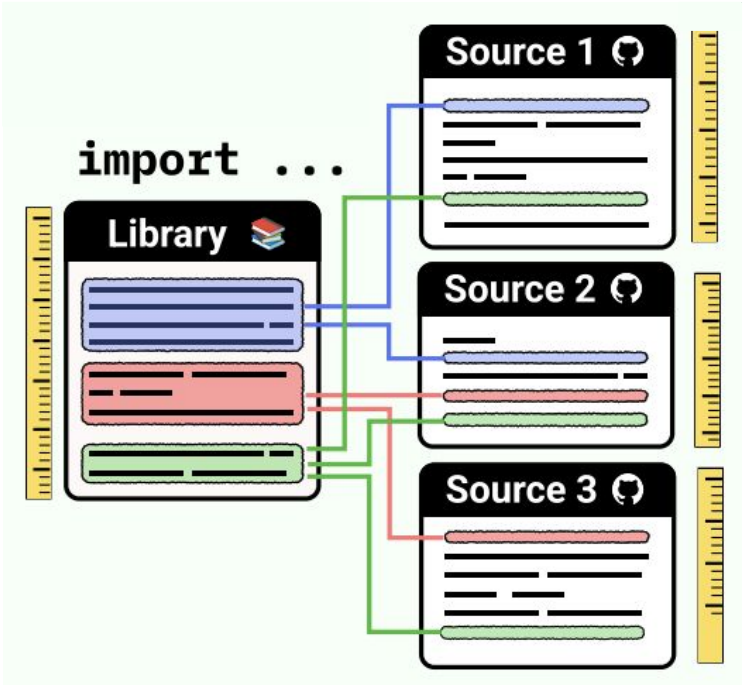


A new library

$$\mathcal{L}$$

Refactoring of the original source code

$$\{\rho'_n\}_{n=1}^{N}$$



Objective

$$\textbf{Pass Rate} = \frac{\text{Number of unit tests pass}}{\text{All unit tests}}$$

Pass rate of the original source code

$$\forall \rho_n, \tau(\rho_n) \leq \tau(\rho'_n)$$

For each code in the code source

Pass rate of the refactored code

Objective

Loss function

$$\ell\left(\mathcal{L}, \{\rho'_n\}\right) = \begin{cases} -\log p_{\text{LM}}(\mathcal{L}) + \sum_n -\log p_{\text{LM}}(\rho'_n \mid \mathcal{L}) & \forall \rho_n, \tau(\rho_n) \leq \tau(\rho'_n) \\ \infty & \text{otherwise} \end{cases}$$

Refactorings that pass at least as many tests as the original program

Objective

Loss function

Refactorings that pass at least as many tests as the original program

$$\ell\left(\mathcal{L}, \{\rho'_n\}\right) = \begin{cases} -\log p_{\text{LM}}(\mathcal{L}) + \sum_n -\log p_{\text{LM}}(\rho'_n \mid \mathcal{L}) & \forall \rho_n, \tau(\rho_n) \leq \tau(\rho'_n) \\ \infty & \text{otherwise} \end{cases}$$

Objective

Loss function

Prob to get the library under a language model

Given the library, the prob to get the refactored program

Refactorings that pass at least as many tests as the original program

$$\ell\left(\mathcal{L}, \{\rho'_n\}\right) = \begin{cases} -\log p_{\mathrm{LM}}(\mathcal{L}) + \sum_n -\log p_{\mathrm{LM}}(\rho'_n \mid \mathcal{L}) & \forall \rho_n, \tau(\rho_n) \leq \tau(\rho'_n) \\ \infty & \text{otherwise} \end{cases}$$

Method

LIBRARIAN: Refactoring Code to Create Libraries

Minimize the loss function

$$\mathcal{L}^\star, \{\rho_n^\star\} = \underset{\mathcal{L}, \{\rho_n'\} \in \textsc{Sample}(\{\rho_n\})}{\arg\min} \ell\left(\mathcal{L}, \{\rho_n'\}\right).$$

Groups of programs that likely share structures that can be abstracted into general library functions

Method

LIBRARIAN: Refactoring Code to Create Libraries

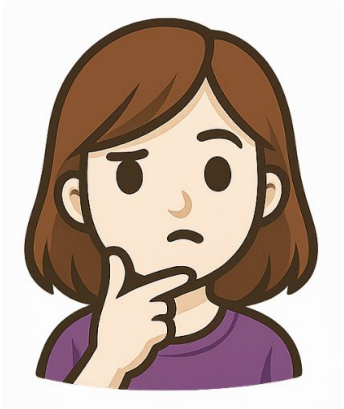
Minimize the loss function

$$\mathcal{L}^\star, \{\rho_n^\star\} = \underset{\mathcal{L}, \{\rho_n'\} \in \textsc{Sample}(\{\rho_n\})}{\arg\min} \ell\left(\mathcal{L}, \{\rho_n'\}\right).$$

Groups of programs that likely share structures that can be abstracted into general library functions

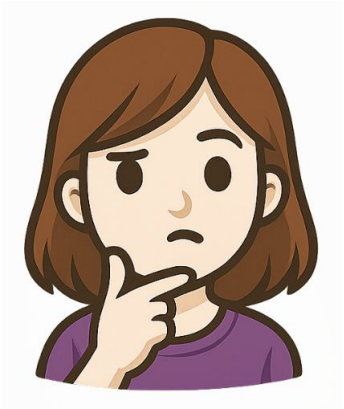
Method

How to find Groups of programs that can be abstracted into general library functions?



Method

How to find Groups of programs that can be abstracted into general library functions?



Sample with clustering

Method

How to find Groups of programs that can be abstracted into general library functions?

ReGAL

1. Embedding the queries
2. Clustering similar queries
3. Sorting them by token length (Complexity)

Method

How to find Groups of programs that can be abstracted into general library functions?

ReGAL

1. Embedding the queries
2. Clustering similar queries
3. Sorting them by token length (Complexity)

LIBRARIAN

1. Summarize the code using LLM
2. Embedding the summaries
3. Clustering similar Summaries

Method

How to find Groups of programs that can be abstracted into general library functions?

ReGAL

1. Embedding the queries
2. Clustering similar queries
3. Sorting them by token length (Complexity)

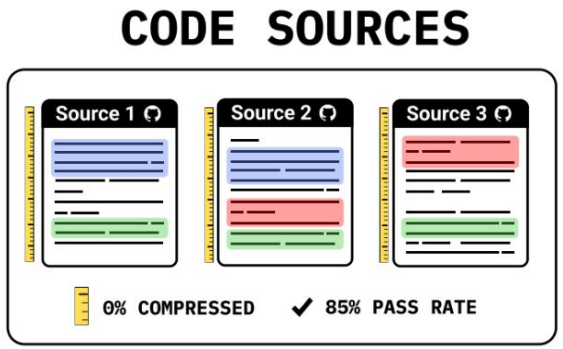
Query “Sort a list” would cluster together, even if one program is implementing quicksort and one bubble sort.

LIBRARIAN

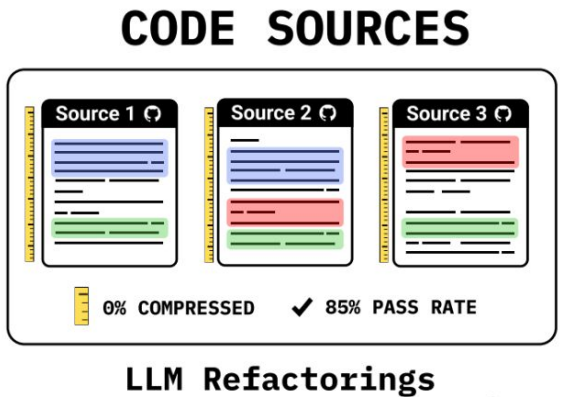
1. Summarize the code using LLM
2. Embedding the summaries
3. Clustering similar Summaries

Two programs implement nested loops for pairwise comparison, can be clustered together, even if one is “sort a list” and the other is “find duplicate elements”.

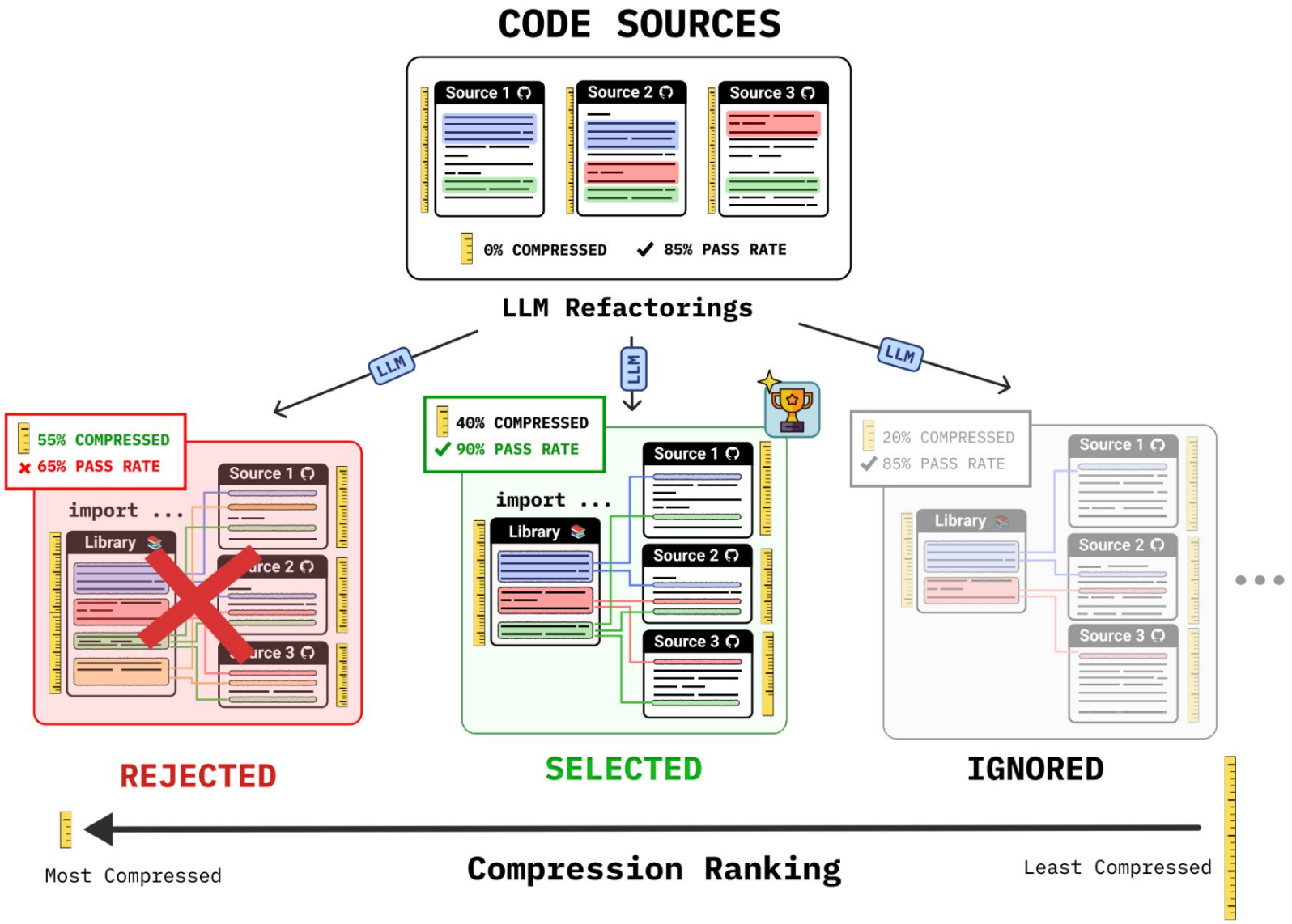
Librarian - Clustering



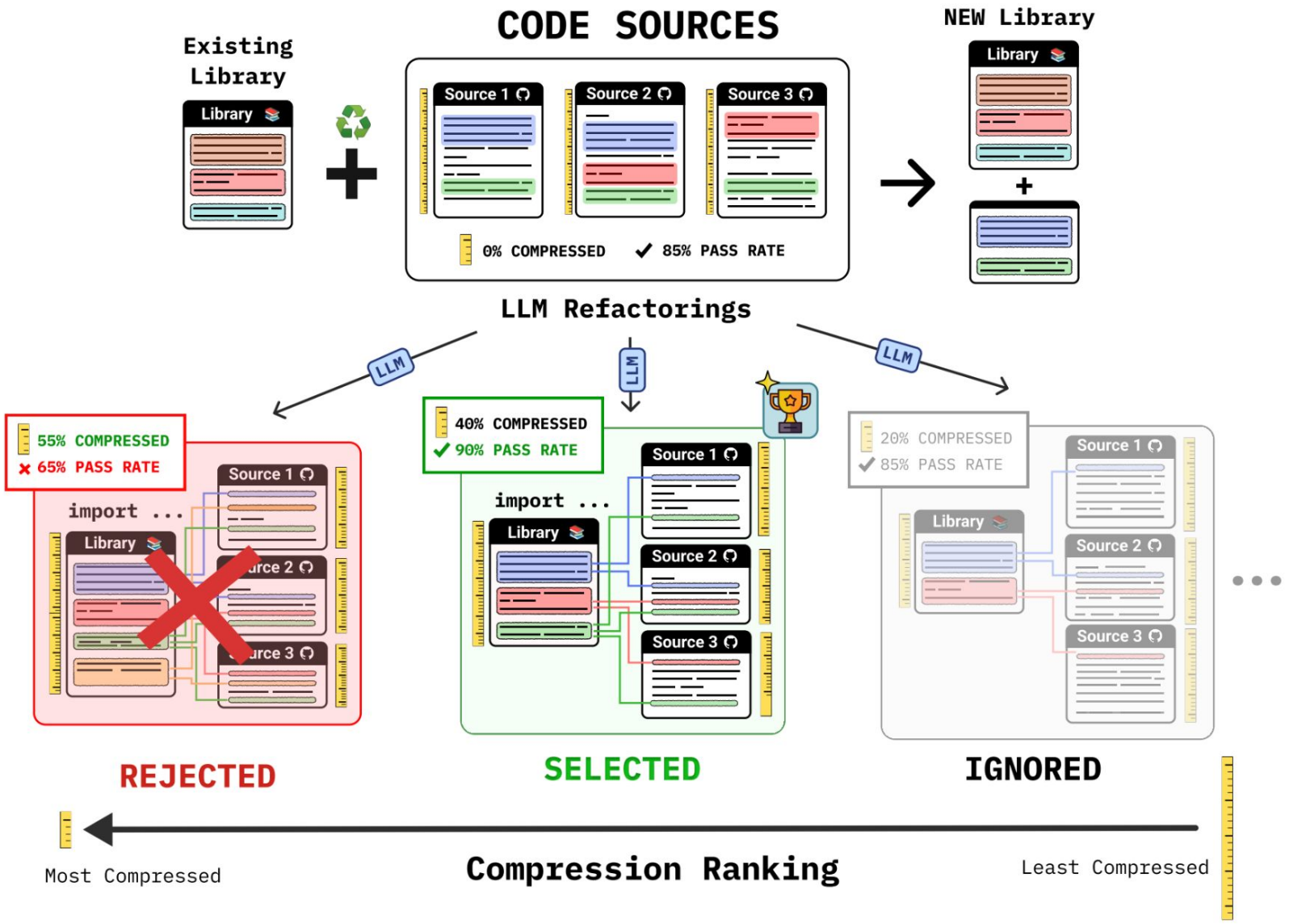
Librarian - Sampling Refactoring



Librarian - Ranking with Compression



Librarian - Full pipeline



The MiniCode Benchmark

A new benchmark where code agents must minimize and refactor multiple independent solutions into a joint library.

- Agents must create a library that can be imported into multiple repositories.
- Complete freedom to design the interface and implementation.
- Unit tests from all repositories that will import the designed library.
- Agents must understand and synthesize information from multiple repositories to design a unified library.

The MiniCode Benchmark

Table 1: MINICODE Statisics

Domain	Sources	Collections	Avg LoC	Avg Tests	Gen by
Code Contests	300	30	87	10	Humans
Small Repositories	262	22	209	12	o4-mini
Large Repositories	20	10	6,433	101	Claude-Sonnet 3.7

Evaluation

Table 3: Solving downstream program synthesis tasks using learned libraries

Dataset	Model	Pass Rate
Logo	REGAL (gpt-3.5-turbo)	49.3% ±1.1
	LIBRARIAN (3.5-turbo)	69.9% ±0.9
Date	REGAL (gpt-3.5-turbo)	90.2% ±0.5
	LIBRARIAN (3.5-turbo)	94.7% ±0.7

Evaluation

Table 2: Results on the MiniCode CodeContests split

Agent	Pass %	MDL %
original	82.0	100.0
sonnet 3.7	93.9	107.4
sonnet 4	84.4	77.1
codex-mini	82.0	86.8
LIBRARIAN	90.67	53.0

We also present the results on the small repo split, which consists of repositories generated by o4-mini. We experiment with Claude Sonnet 3.7 as a planner, and either Sonnet 3.7 as the implementer or o4-mini.

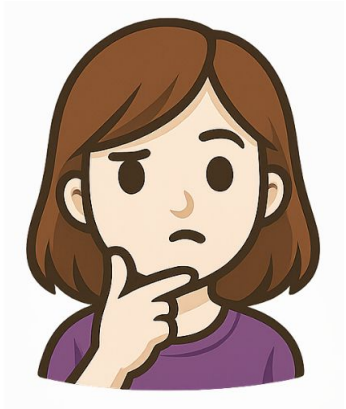
Evaluation

Table 3: Average results on MiniCode-repositories small

Agent	Pass rate (%)	MDL ratio (%)
original	100	100
Claude Sonnet 3.7 planner, Claude Sonnet 3.7 implementer	78	514
Claude Sonnet 3.7 planner, o4-mini implementer	80	244

Finally, we present results on the large repo split. Due to the stronger performance of Sonnet models, we evaluate only Sonnet models to minimize cost.

The end (Or just the beginning)



?