Institute for Data Science & Informatics
University of Missouri

College of Engineering
University of Missouri

2025 AGI Mizzou Hackathon

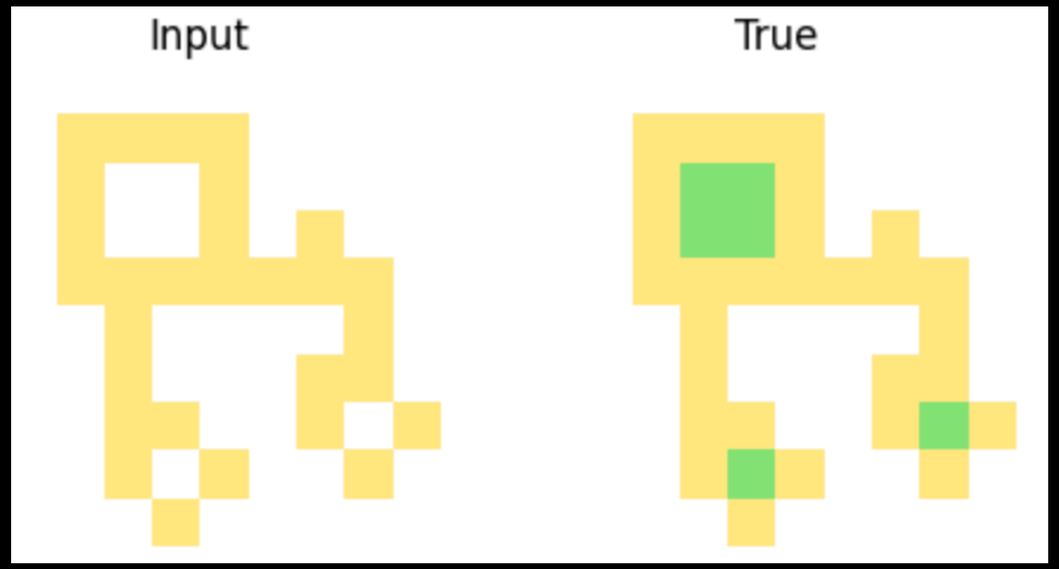
Team t[AI]ger roar

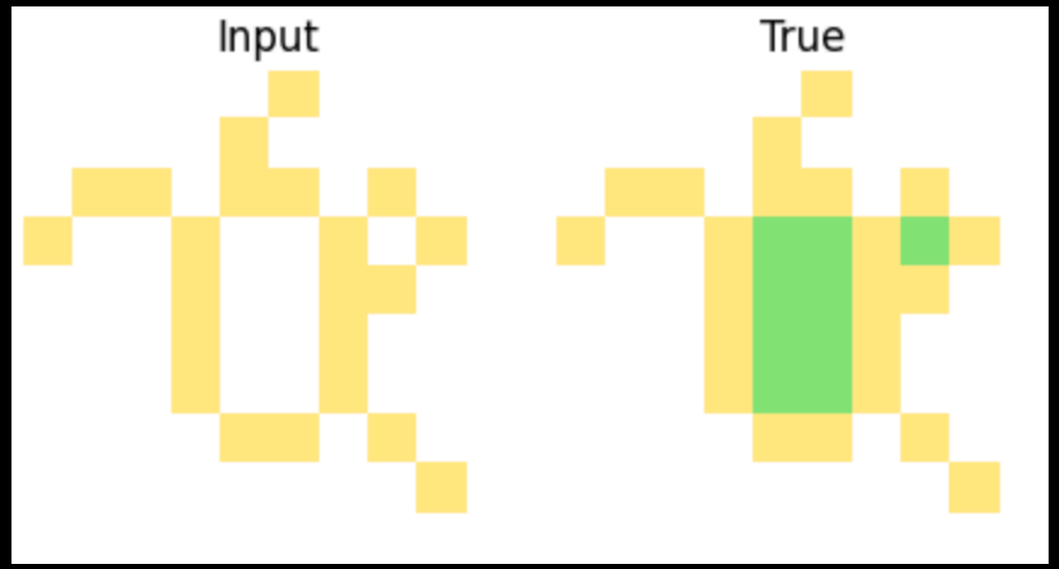
Wen-Hsin Chen | Jeong Wook Lee | Alina Rohulia | Samrat Kumar Dey | Kun-Yi Chen

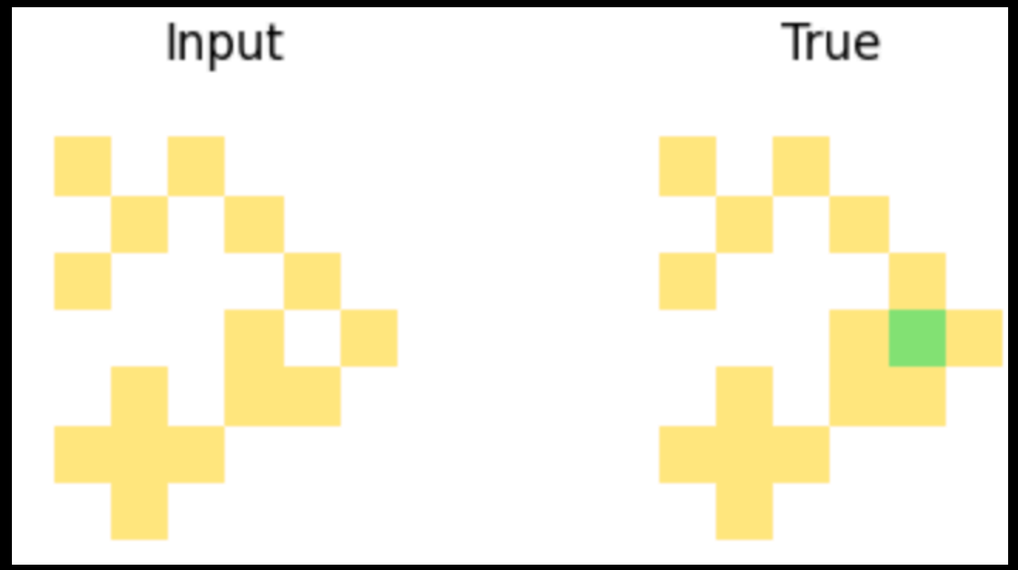
What is ARC-AGI?

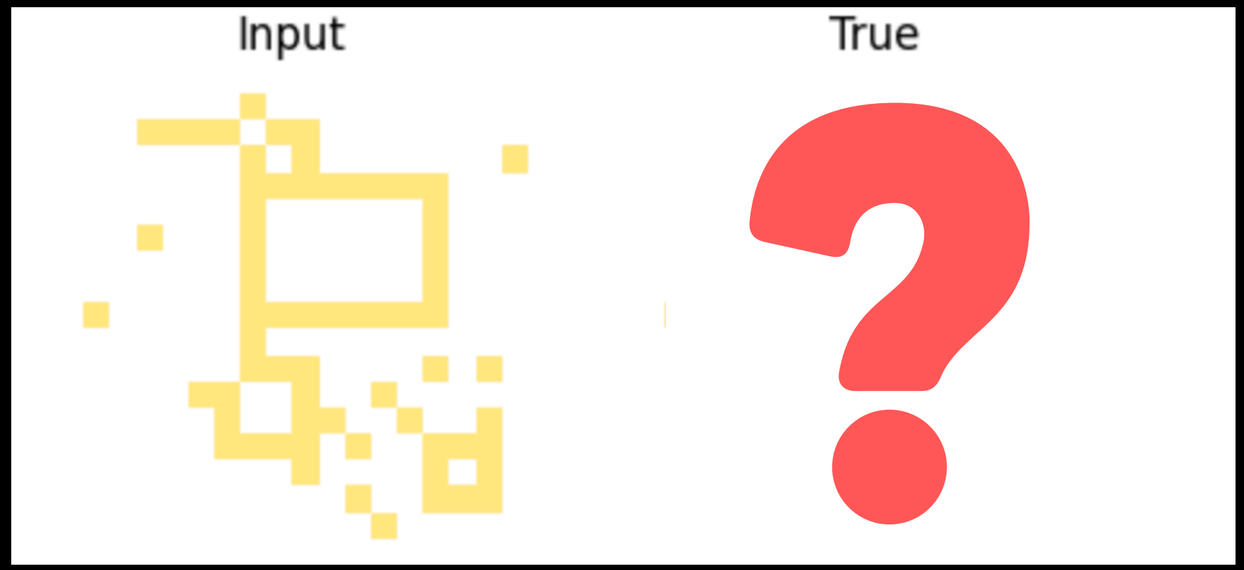
“It’s easy for humans, but hard for AI.”

Human solving...

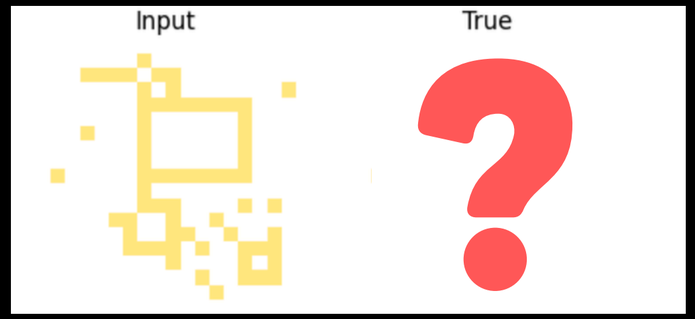


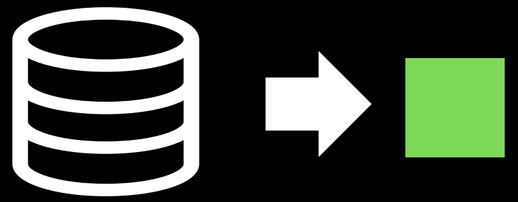




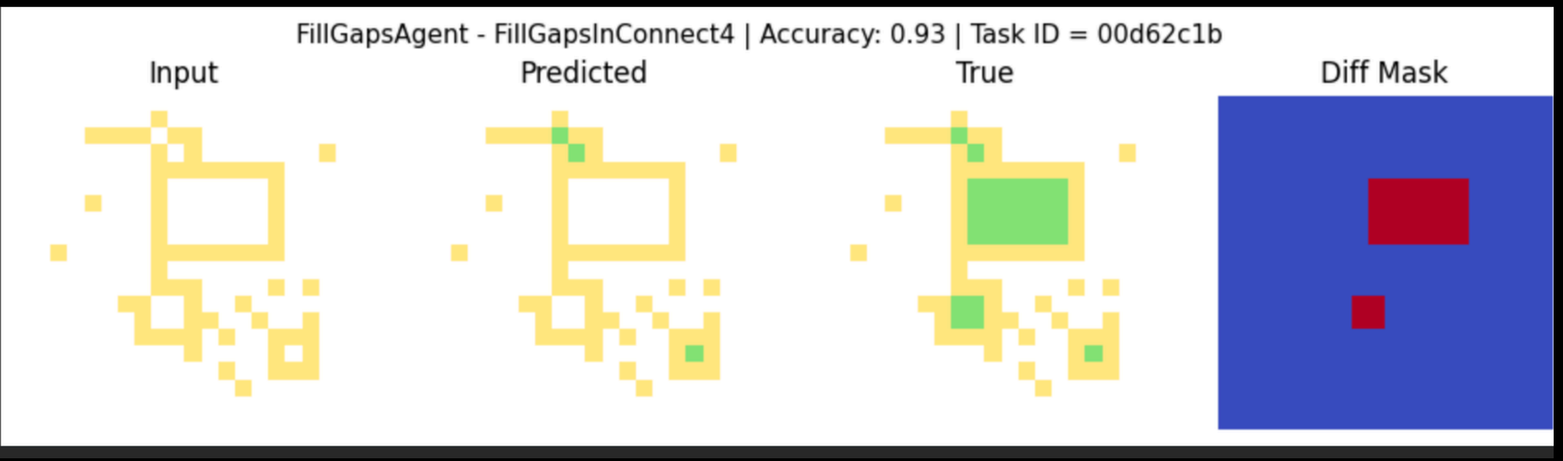


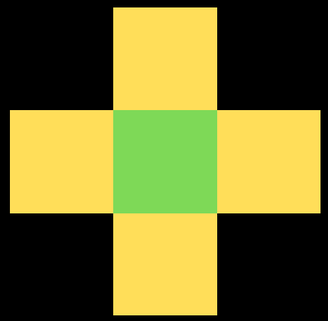
Machine solving...



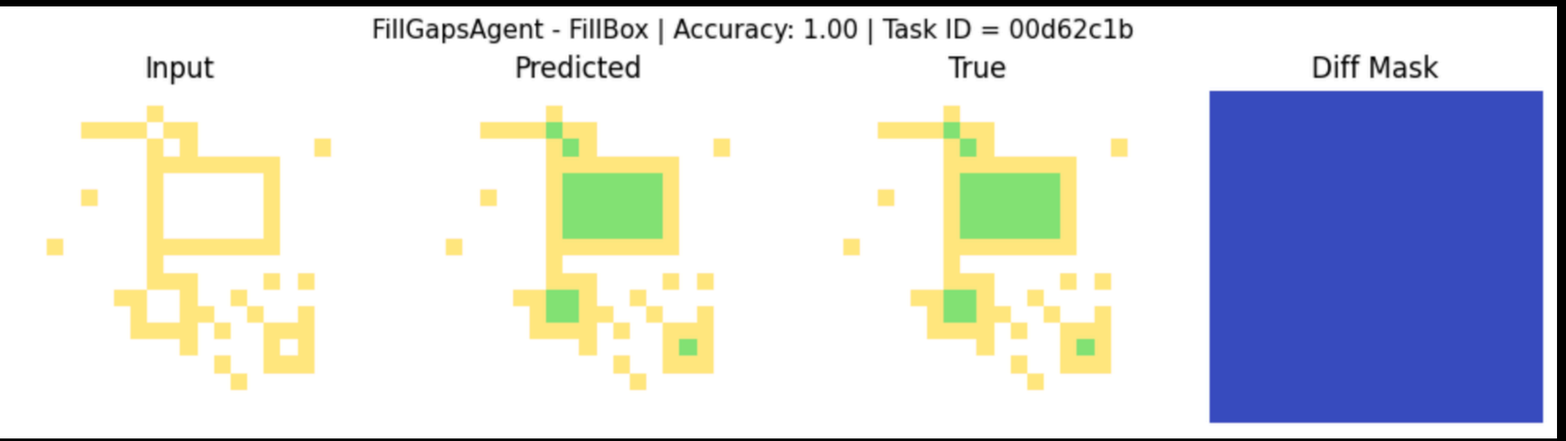


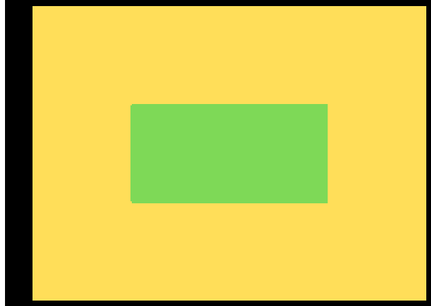
Fetch learned fill color: green





Fill in when ←,↑,→,↓ are all yellow

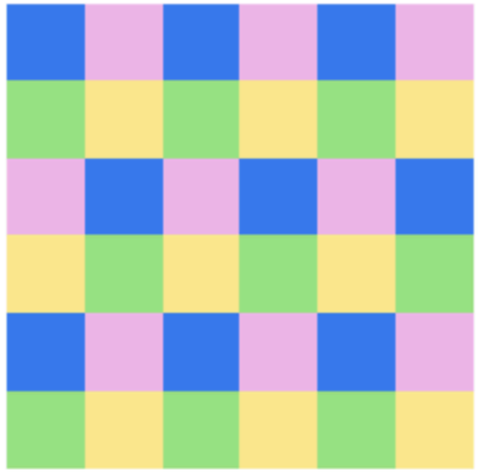




Fill in when box borders are all yellow

Input

True



Input

True

and 1000+ unique problems!

Problem 1

How to increase number of unique problems solved?

Problem 2

How to develop Artificial General Intelligence (AGI)?

How to do this as a student?

Solution Cost

$0

System Configuration

Mizzou HellBender

Node type: GPU (H100)

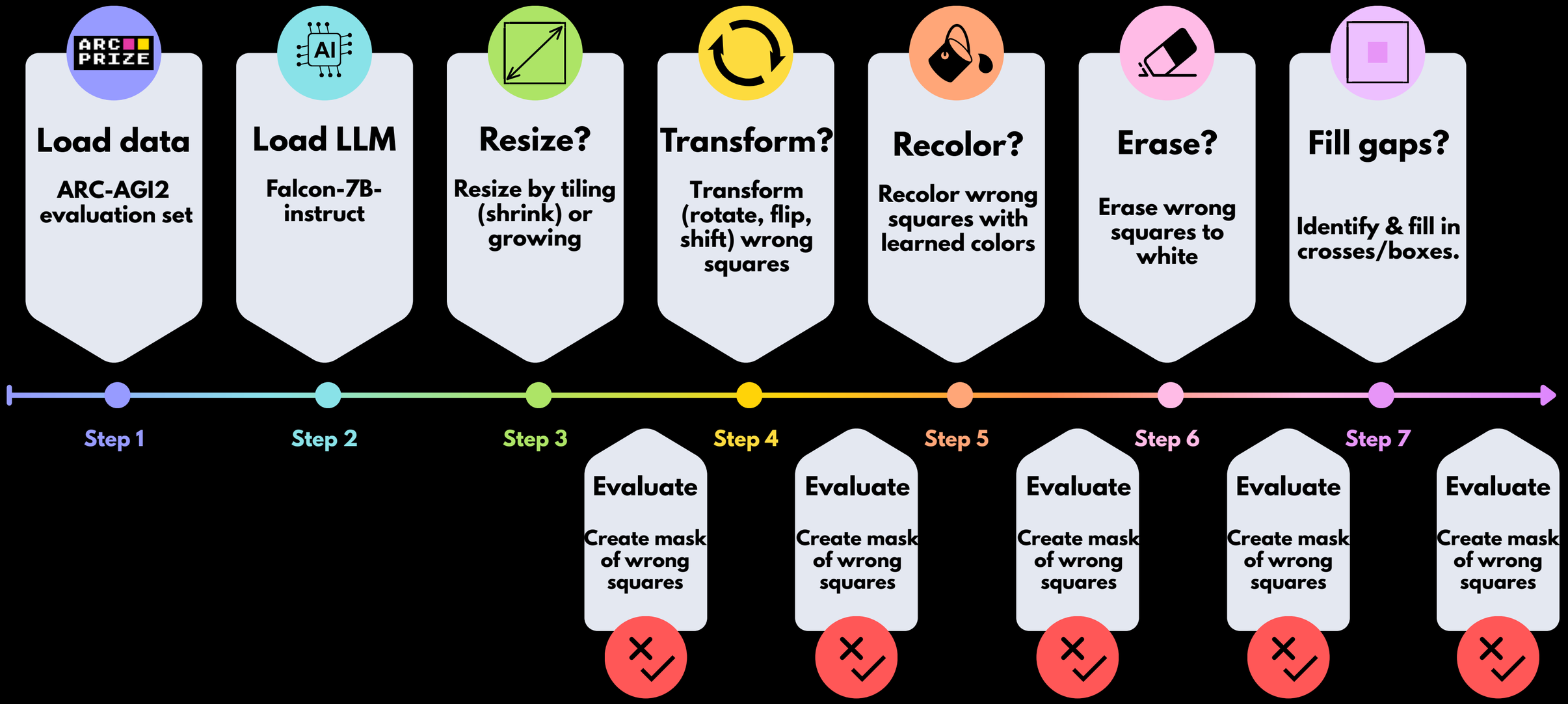
Number of CPUs: 24

Memory: 32 GB

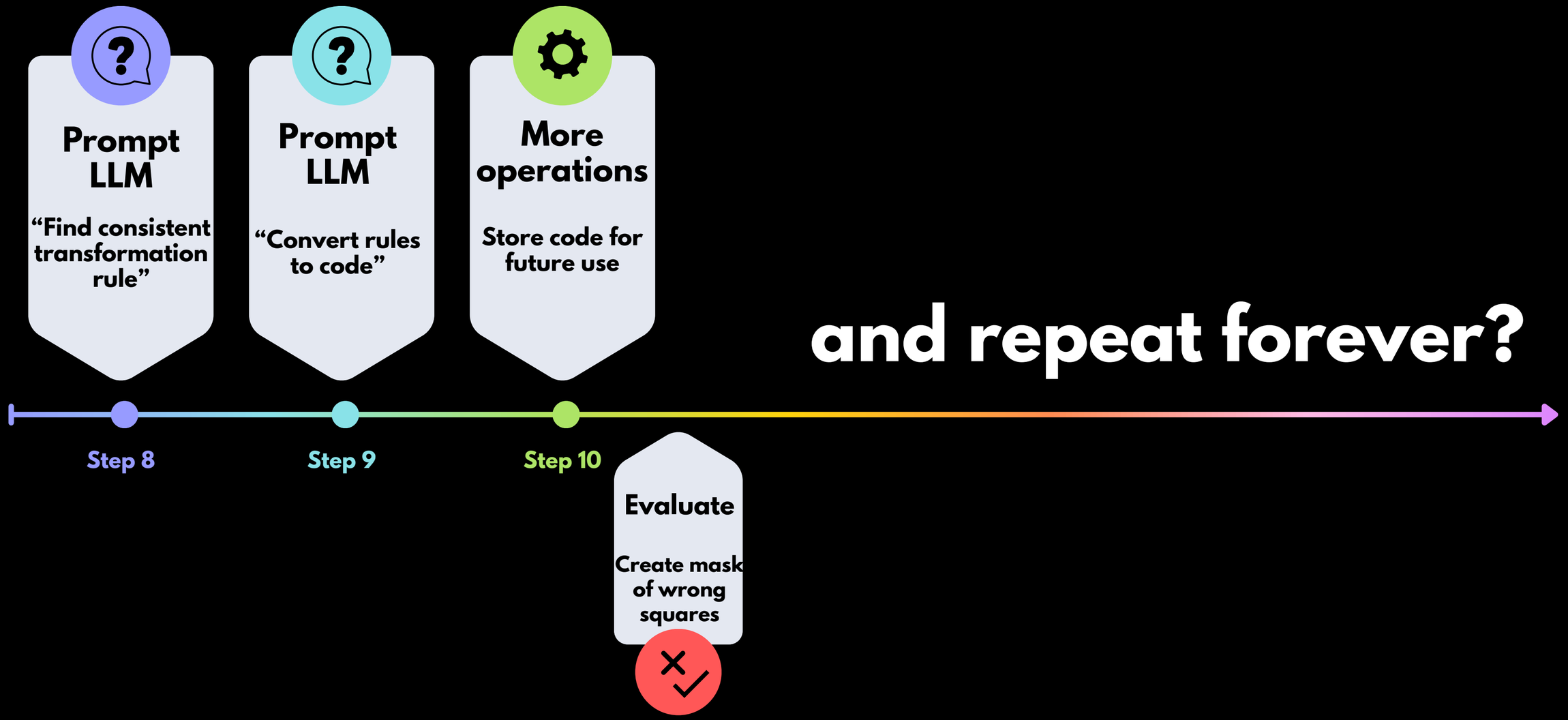
Question 1

How to increase number of unique problems solved?

Methodology



Methodology



LLM setup

Falcon-7B (H100 Optimized)

↳ 1 cell hidden

Operations setup

Load Resize, Transforms, Recolor, Erase, FillGaps, etc.

↳ 7 cells hidden

Execute methodology

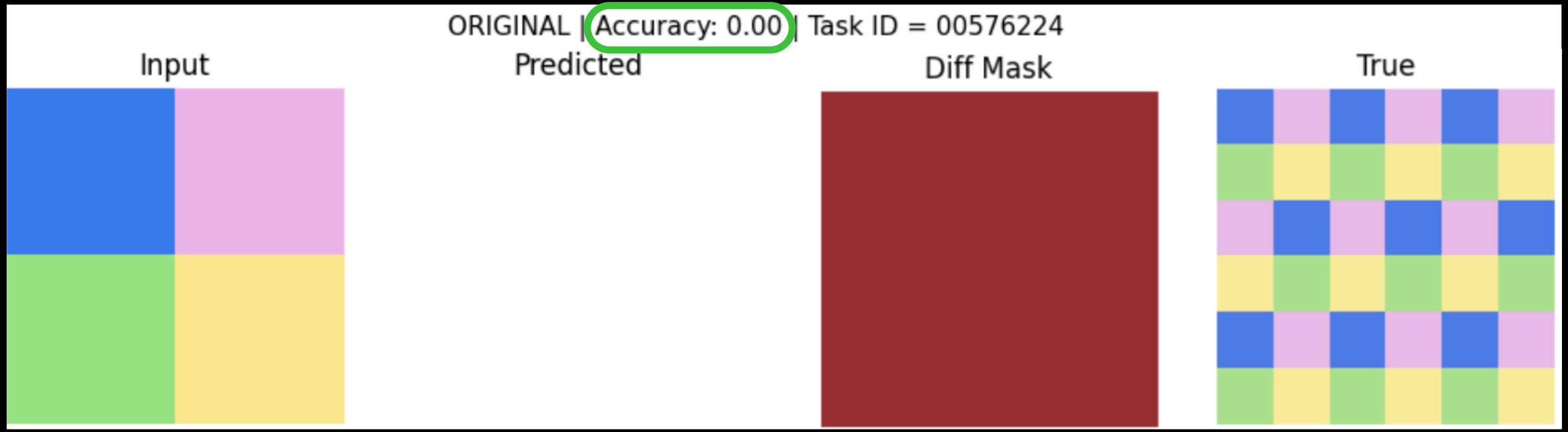
[11] 57s

```
# Lets try first few examples
start = 0
stop = 6
x= 0
test_one = False
for id in train_ids[start:stop]:
# Fetch test from train
    test_arr, label_arr = train_data[id]["test"][0]["input"], train_label[id][0]
    test_arr, label_arr = convert_2D_numpy(test_arr), convert_2D_numpy(label_arr)

# Define Agents
```

~1 min to solve first 6 unique problems

Play slide 14 video


ORIGINAL | Accuracy: 0.00 | Task ID = 00576224
Input
Predicted
Diff Mask
True

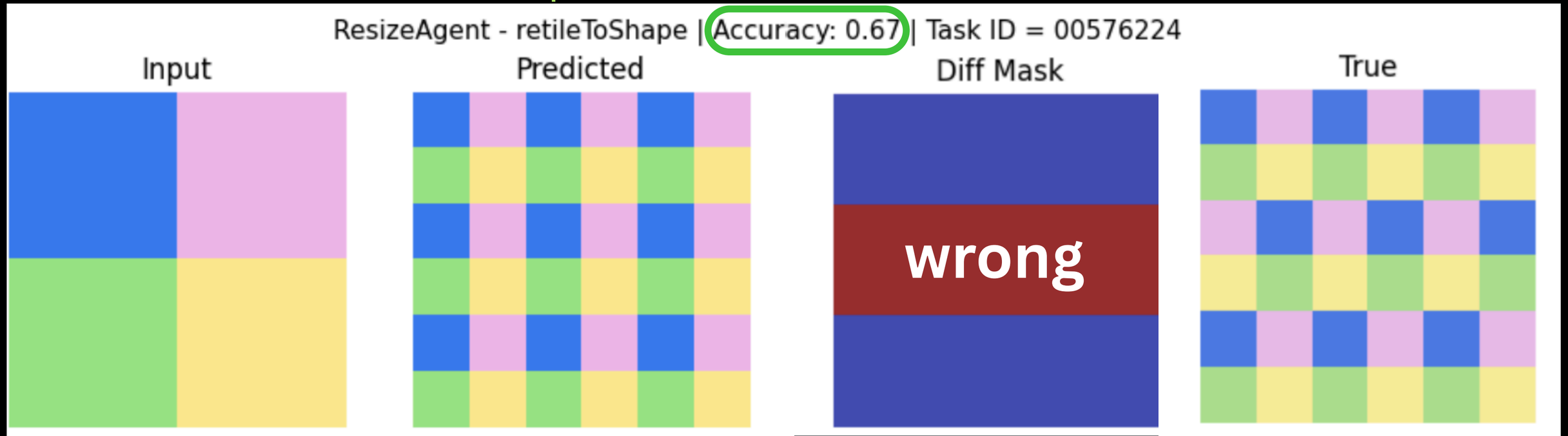
Resize?

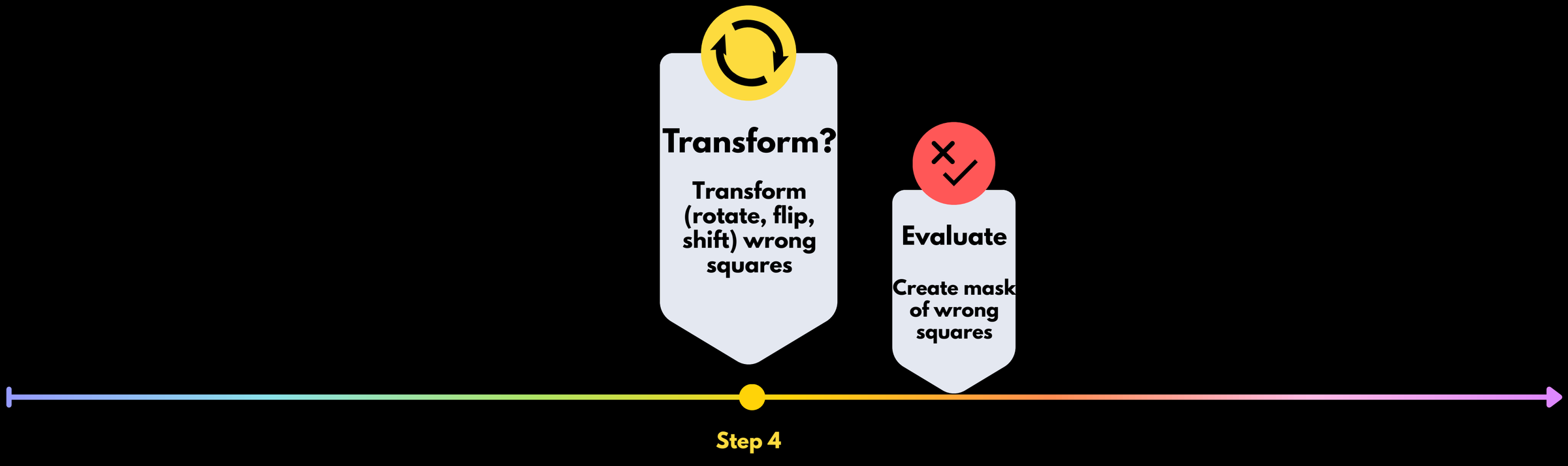
Resize by tiling (shrink) or growing

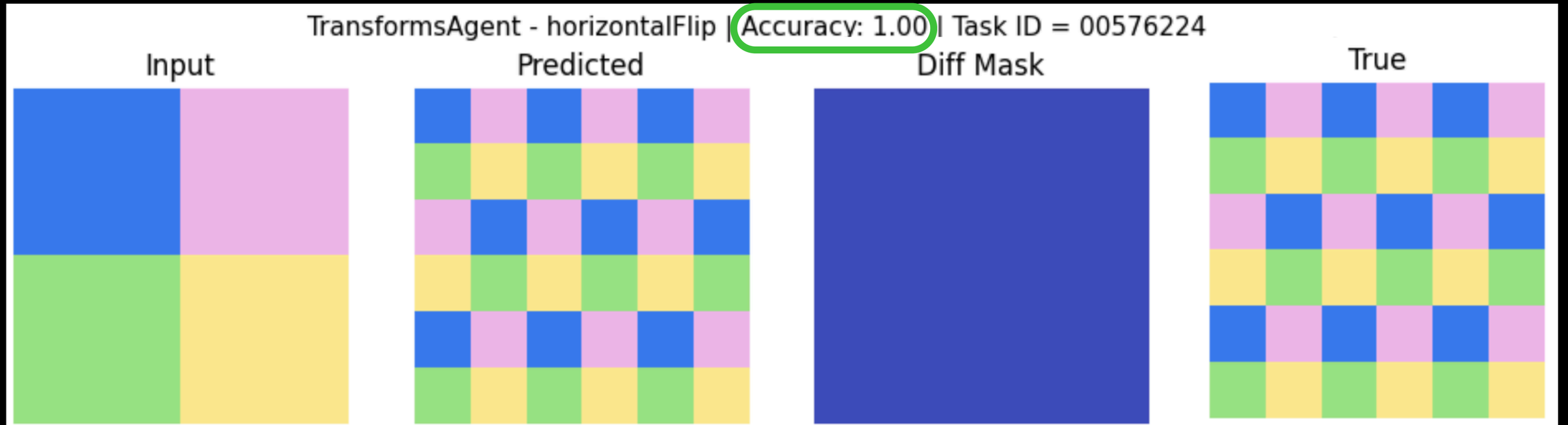
Evaluate

Create mask of wrong squares

Step 3



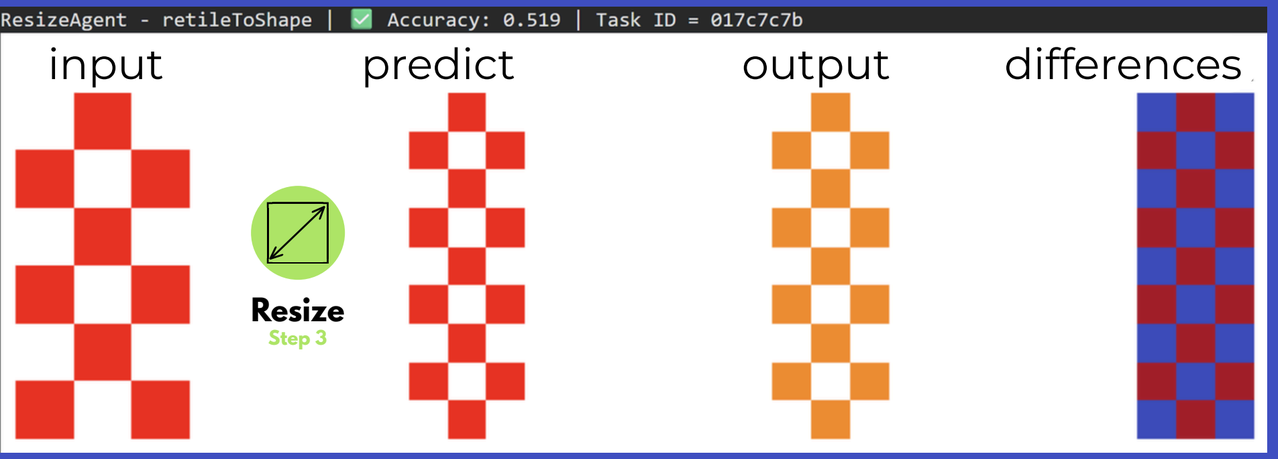

Transform?
Transform (rotate, flip, shift) wrong squares
Evaluate
Create mask of wrong squares
Step 4

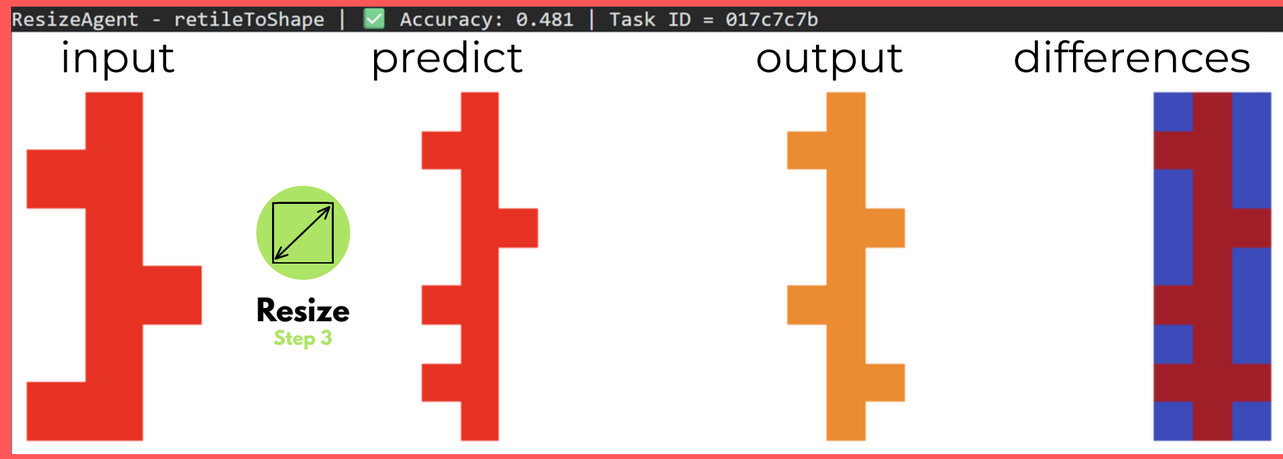

TransformsAgent - horizontalFlip | Accuracy: 1.00 | Task ID = 00576224
Input
Predicted
Diff Mask
True

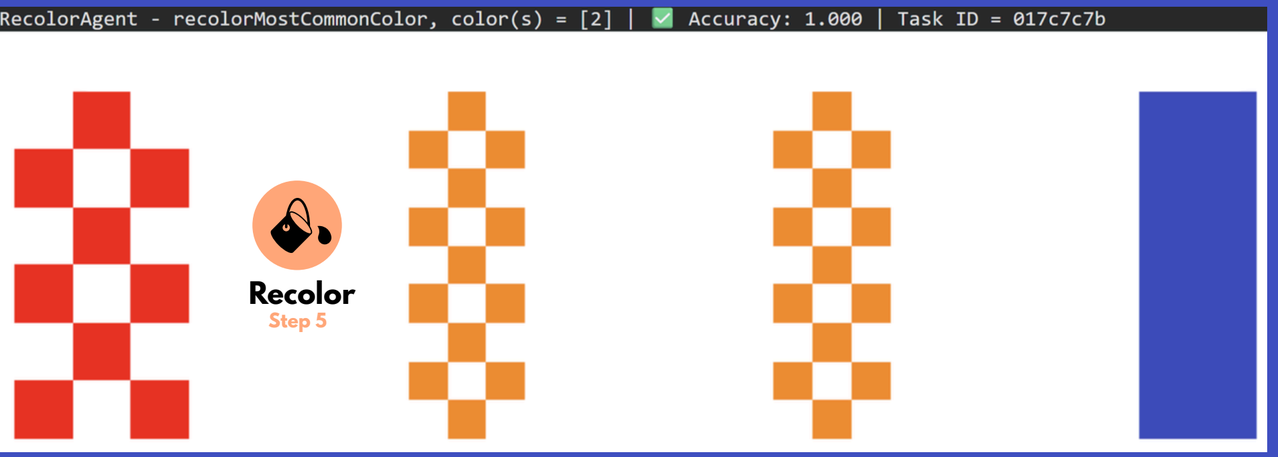
Results on evaluation problems

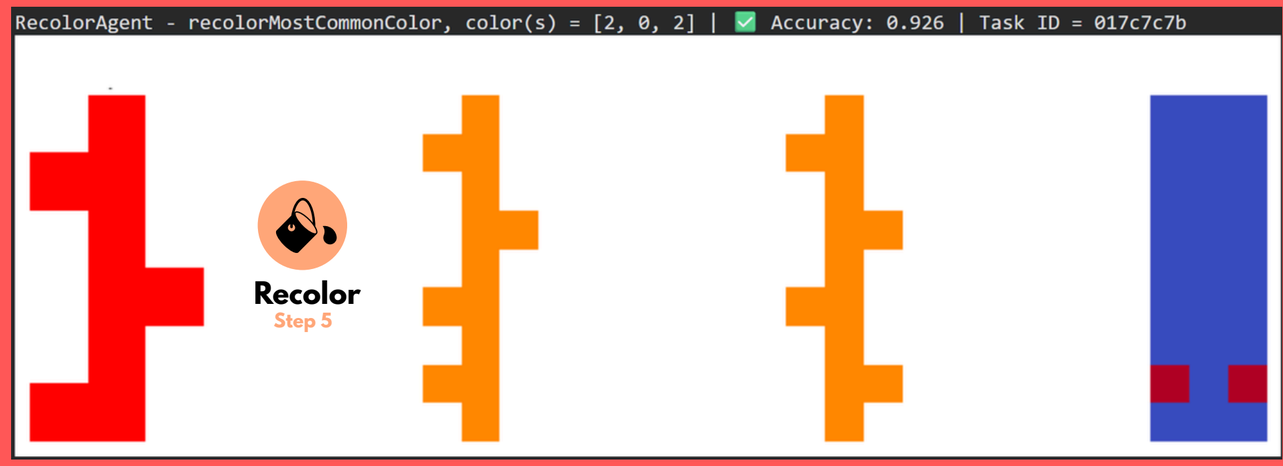
Pixel Correctness (Fine-Grained Accuracy)	Correct / Incorrect (Task-Level Accuracy)
91.23%	**10.59%**
across each example	120 problems total

Even within same problem, operations can vary



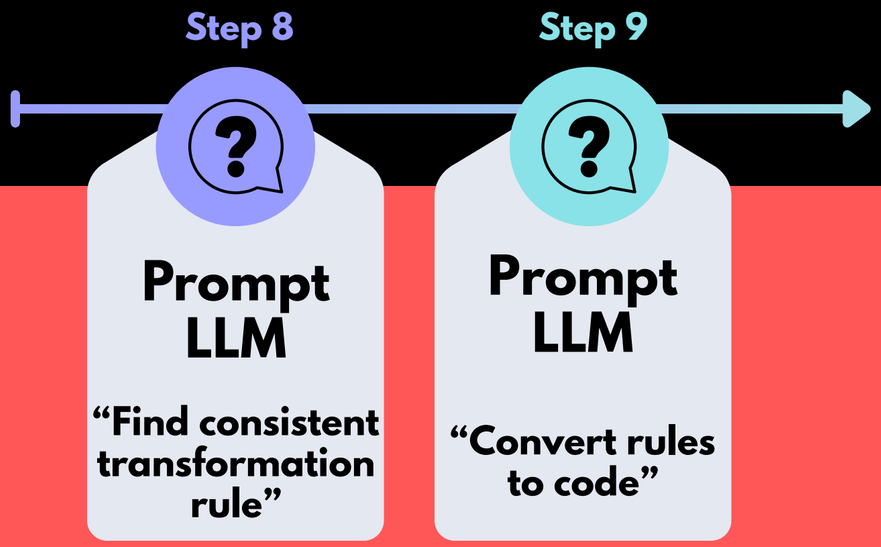




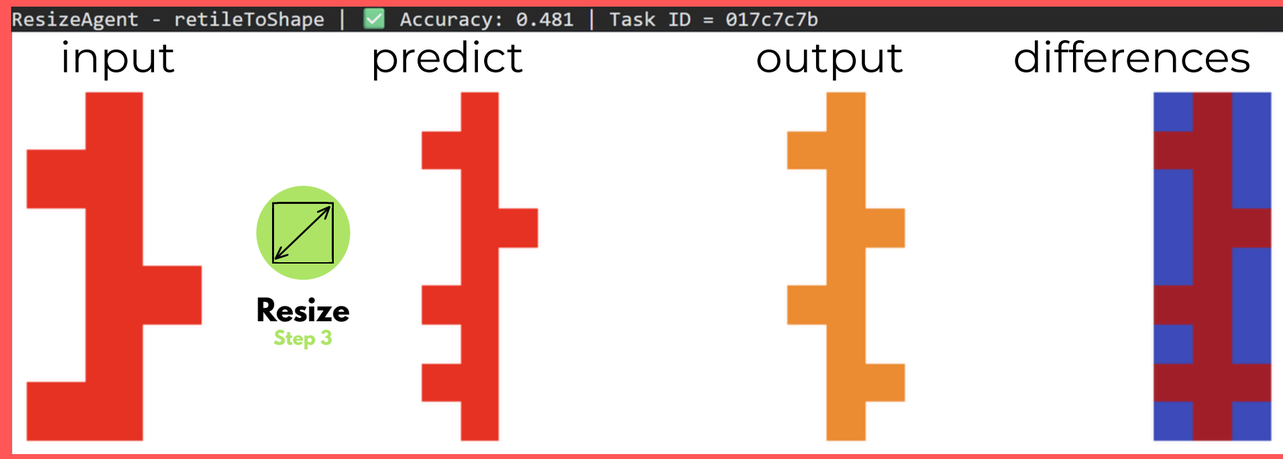


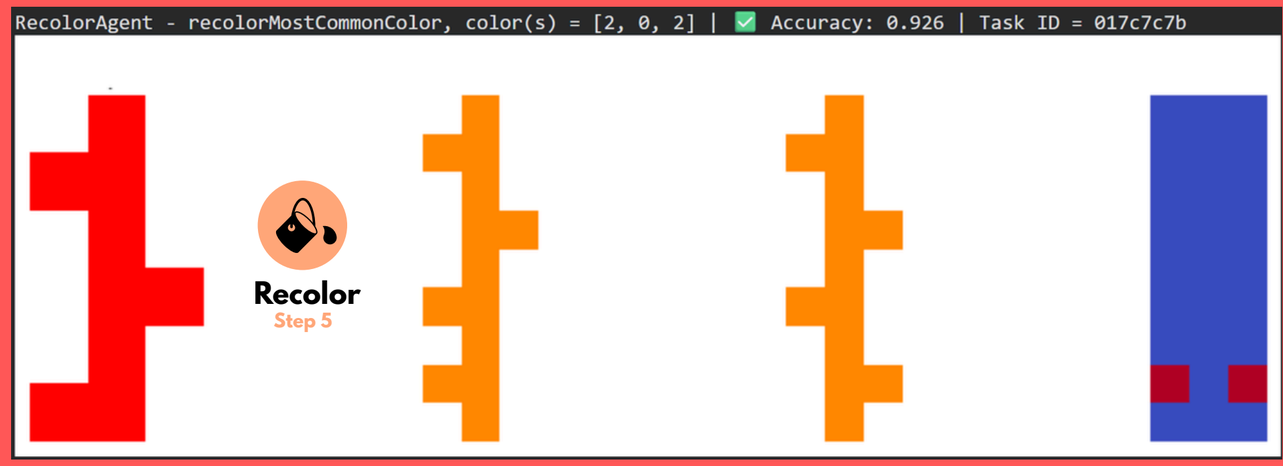
Correct

Incorrect



prompt = “Example1: Agents can't solve task. Best accuracy=0.93 using operations [{'RecolorAgent': ['recolorMostCommonColor, color(s) = [2, 0, 2]']}].
best_predict_arr=[[0 2 0]...
input_arr=[[0 1 0]...
true_arr=[[0 2 0]...
differences=[[False False False]... [True False True]...
1. Identify what is still different
2. List the reasoning behind input_arr -> true_arr?
3. Generalize reasoning so it'll work for other similar examples
4. Generate python code that would solve this task”





Incorrect ✗

Question 2

How to develop Artificial General Intelligence (AGI)?

Methodology for AGI

Created primitive operations

Store operations for future use

Step 10

Evaluate

Create mask of wrong squares

Outside help for LLM?

RAG on Numpy libraries for context

Step 11

Evaluate

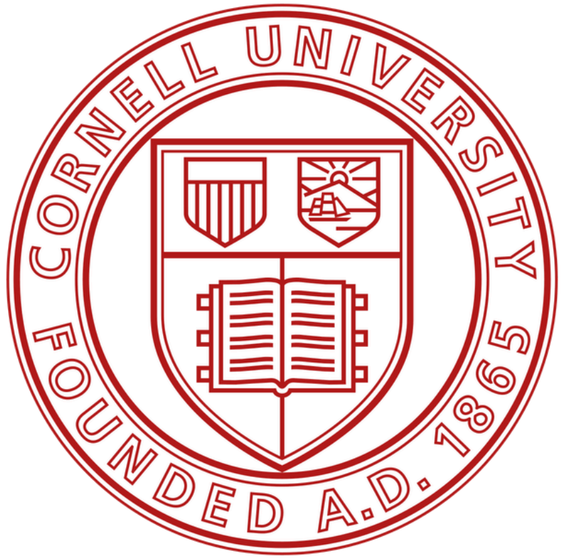
Create mask of wrong squares

Step 12

Use DreamCoder-inspired library learning

Motivation of adapting DreamCoder

- Problems are combinatorial & symbolic (colors, shapes, symmetries, object operations)
- We want compositional generalization: reuse concepts like rotate, crop, repeat across puzzles



Source: https://ml4code.github.io/publications/ellis2021dreamcoder/

Motivation of adapting DreamCoder

Why DreamCoder-style approach?

- System should learn building blocks, not just simply memorize per-task tricks
- A standard neural network outputs a grid directly and is hard to interpret

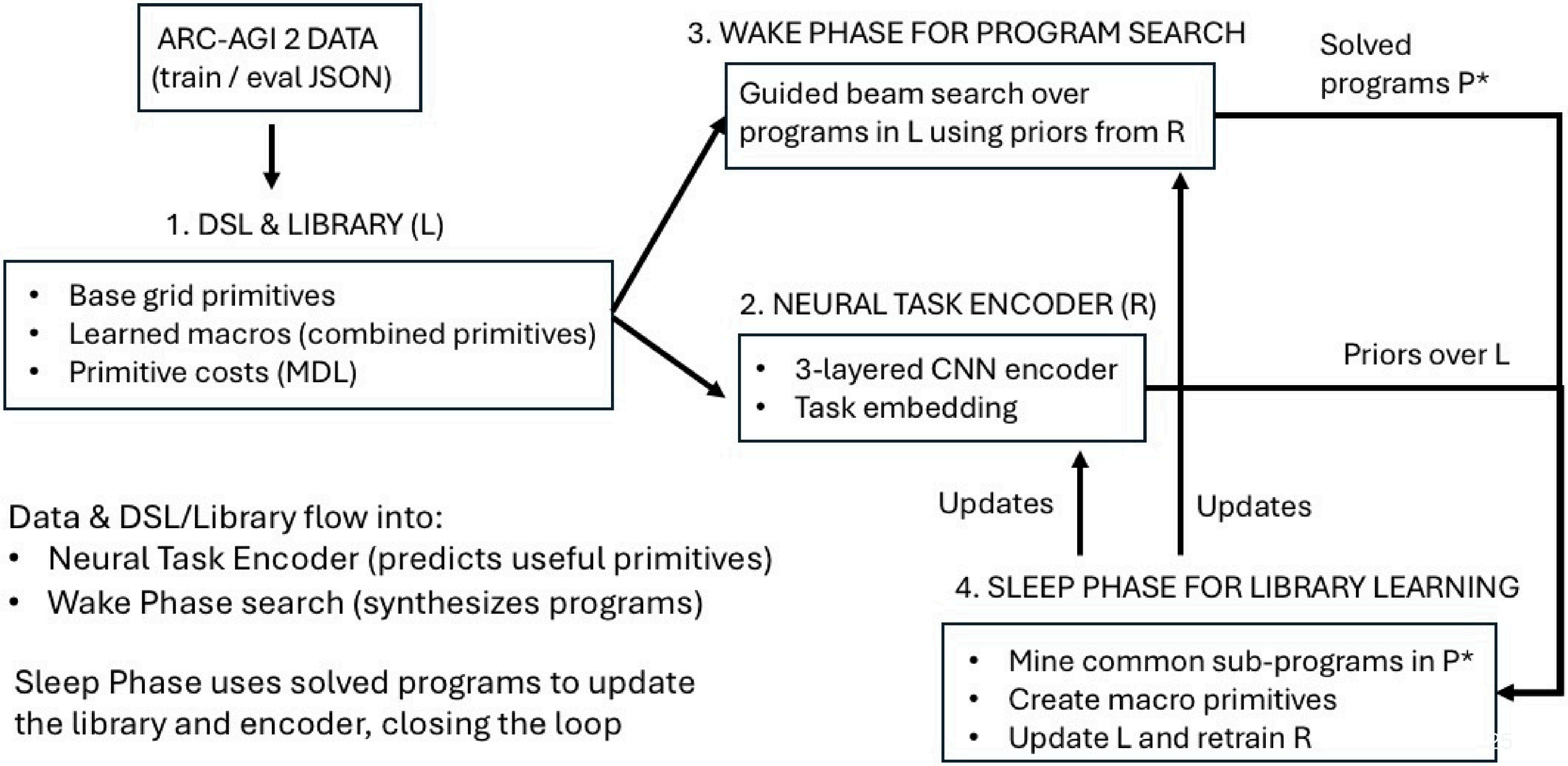
Program-based solutions advantage

- Interpretable (can inspect the synthesized program)
- Reusable (patterns found on one task help on others)

Basic ideas of library learning approach

1. Start with a small DSL
2. Automatically invent higher-level primitives when patterns repeat
3. Judge whether a new primitive is actually useful

Our Architecture Overview



Data & DSL/Library flow into:

- Neural Task Encoder (predicts useful primitives)
- Wake Phase search (synthesizes programs)

Sleep Phase uses solved programs to update the library and encoder, closing the loop

Conclusion

1. **How to increase number of unique problems solved?**
2. **How to develop Artificial General Intelligence (AGI)?**

- Create lots of generalized operations (primatives) through code & LLMs
- Retrieving outside context when needed.
- Abstraction to simplify future problem solving.

At every step, evaluate! This informs model which squares are still wrong, helping model learn changes in problems & better adapt

Impact
(beyond solving squares)

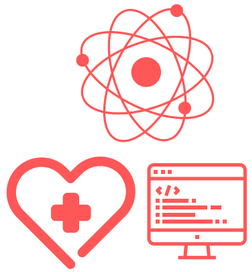
The future for AI is human-reasoning.

Currently, even “PhD-level” AI struggles with tasks outside their trained probability distributions
AI lacks human reasoning skills. Thus, our methodology mimics the best of human reasoning--

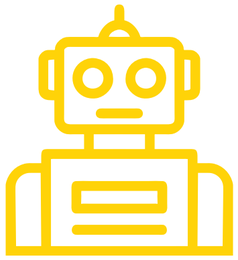
The scientific method:

1. Analyze data & solve empirically - Code primatives
2. Retrieve help from outside - LLM and RAG for context
3. Abstract and generalize - DreamCoder
4. Adapt to changes based on evaluation - Evaluation at every step

This is how AI can advance to reasoning and general intelligence



Domain-specific applications



AI

Scientific method

Questions?

Thank you!

DreamCoder details

Architecture part 1: DSL & Library (L) –The Building Block

Overview

- Each primitive is implemented as a Python function (mirror, rotate, recolor)
- Calculate cost (how many primitives it takes to solve a puzzle)
- Reward short & simple programs

Why?

- Makes search space smaller
- Encourages reuse of general concepts (rotate, crop) instead of task-specific hacks

Architecture part 2: Neural Task Encoder - Guiding Search

Overview

- Small CNN that looks at a puzzle and predicts
- Curveballs
  - Task embedding
  - Probability for each primitive in the library
- Vector of scores over primitives

Why?

- Pure brute-force search over programs is too large
- The encoder acts like a heuristic:
  - “Try these primitives earlier, those later”

Architecture part 3: Wake phase - Program Search

Overview

A guided beam search that tries to build a program

Initialize candidate programs

- Start with simple leaf programs using primitives from the library
- Choose parameters (e.g., target sizes, colors) from observed data (grid sizes, color palette)

Why?

- System synthesizes programs that explain the data

Architecture part-3: Sleep phase

Overview

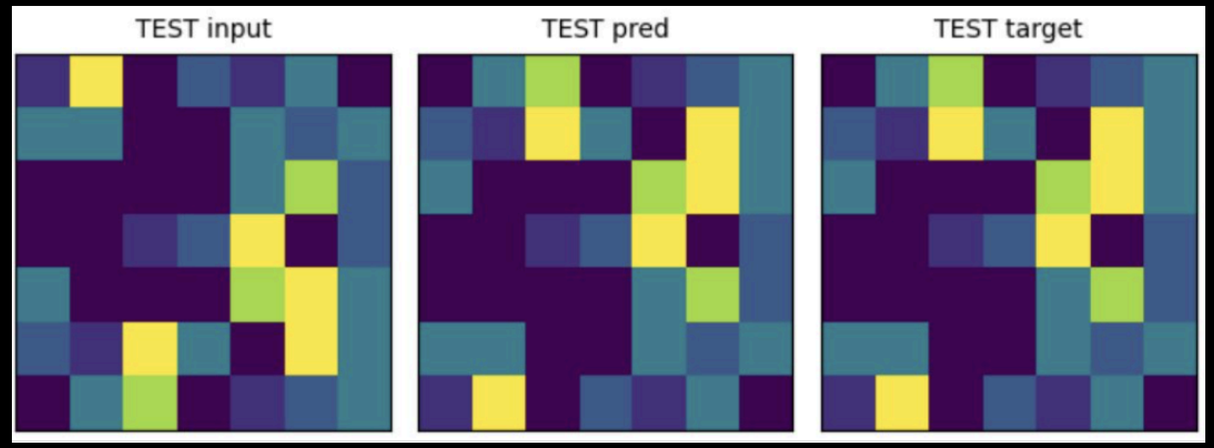
- Collect solved programs and mine frequent sub-programs
- Turn them into new macro primitives with lower Kolmogorov complexity (by program length)
- Update the DSL and retrain the encoder to use these new building blocks

Why?

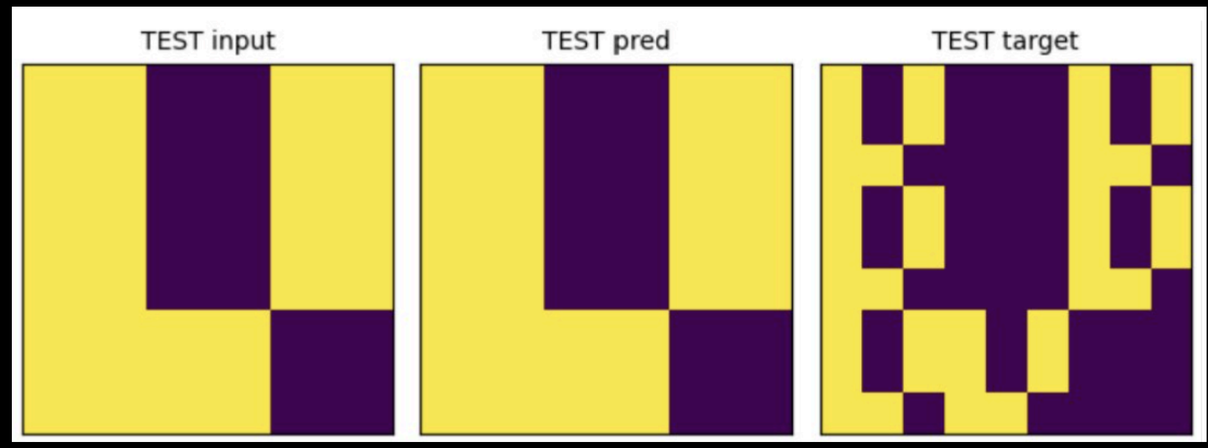
- Over time, the system learns higher-level concepts tailored to the benchmark
- Improves success and learned vocabulary of future search

Results

Correct example



Incorrect example



Among 300 training samples, the library managed to solve only 5 puzzles

College of Engineering
University of Missouri

Institute for
Data Science & Informatics
University of Missouri

Team members





Wen-Hsin Chen	Jeong Wook Lee	Alina Rohulia	Samrat Kumar Dey	Kun-Yi Chen
CS	Informatics	Informatics	Informatics	Informatics