# Solving the ARC Challenge

**Incorporating:**

Broad to Specific Prompting

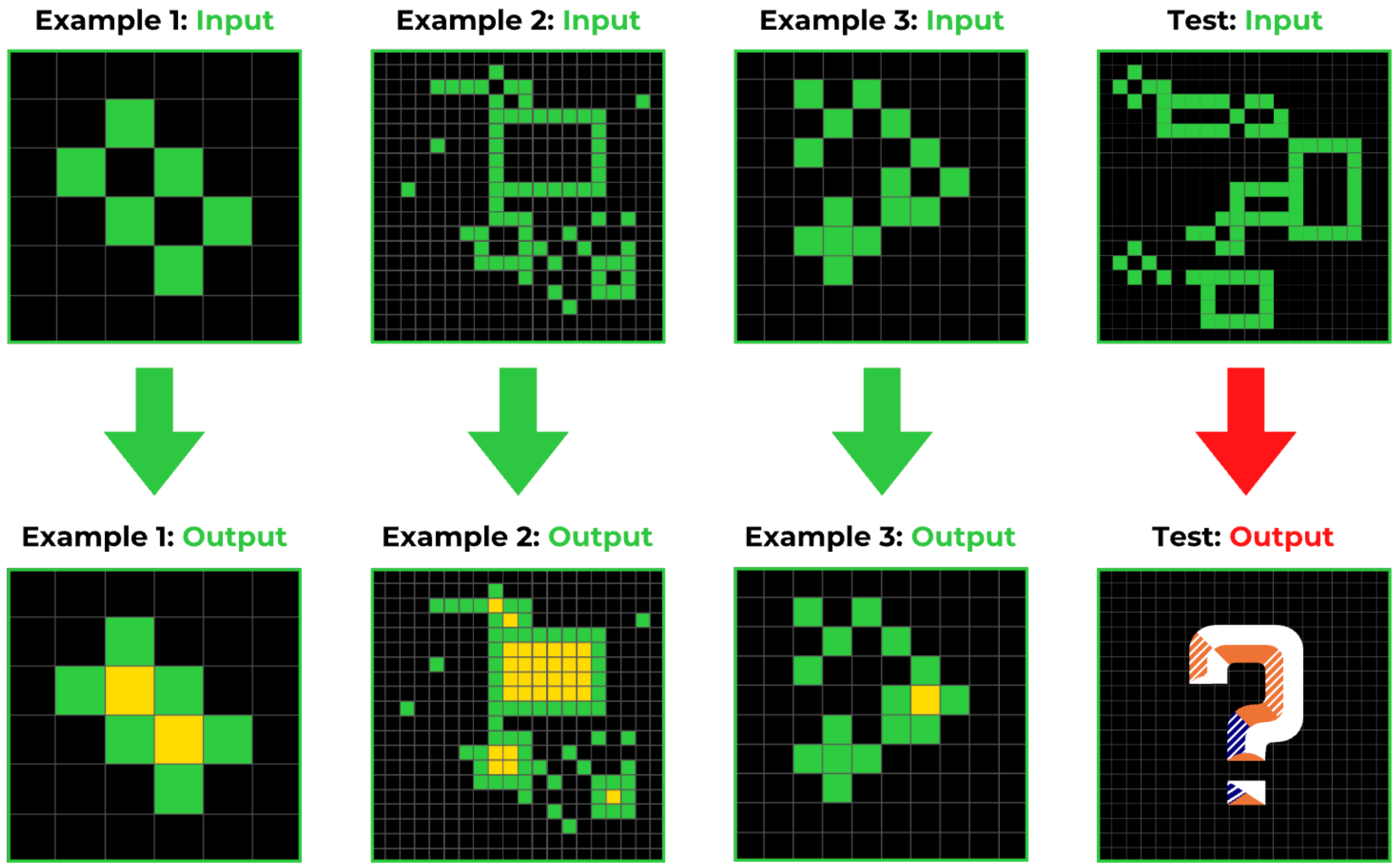
Functional Prompting for Human Priors

Better Image Interpretation

Environment Feedback Loop

**Presented by:**
John Tan Chong Min

# ARC Challenge



## Traditional Methods

- GOFAI, Program Synthesis
  - Needs lots of hand-tuned instructions
- Deep Learning
  - Too many samples needed to learn
- Perhaps another approach is needed

# The Approach: Transformers

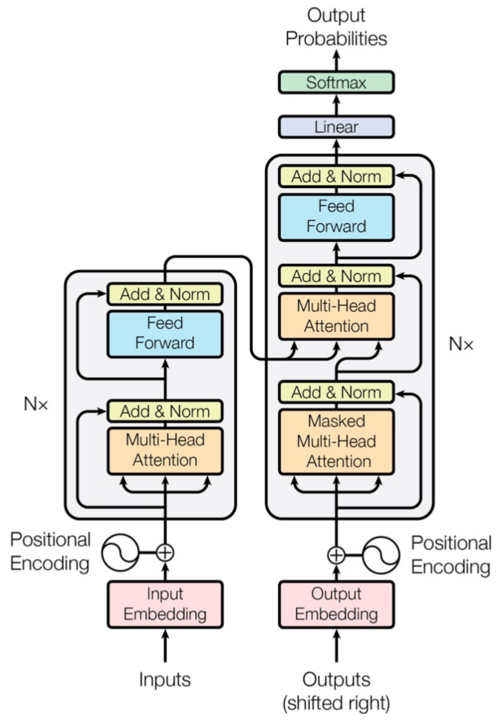


Figure 1: The Transformer - model architecture.

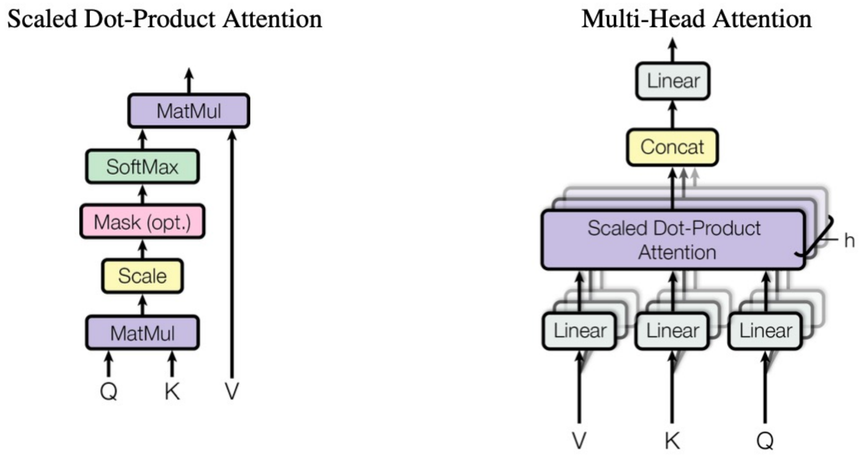


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Taken from: Attention is all you need. Vaswani et al. (2017)

## Zero-shot prompting

- Only applicable if natural language can already describe the categories / use case

- Simply describe the categories / use case in natural text

## Zero-shot prompting

- Example:
  - You are a classification machine. You are to classify the context of each sentence. The various contexts are given as {Context Letter}: {Description}:
    - A: On a mountain
    - B: In the classroom
    - C: In the garden

  - Classify the following and give the answer as {Number}:{Context Letter} for each line. Only provide the context letter without the description:
    - 1. Why, what a steep slope
    - 2. I can’t find my eraser!
    - 3. Those plants are not going to be watering themselves, are they?

## Few-shot prompting

- You are a number classifier. The following are examples of classifications:

    - Input 1: 3
    - Output 1: Odd

    - Input 2: 4
    - Ouput 2: Even

- What is the classification of the number 6?

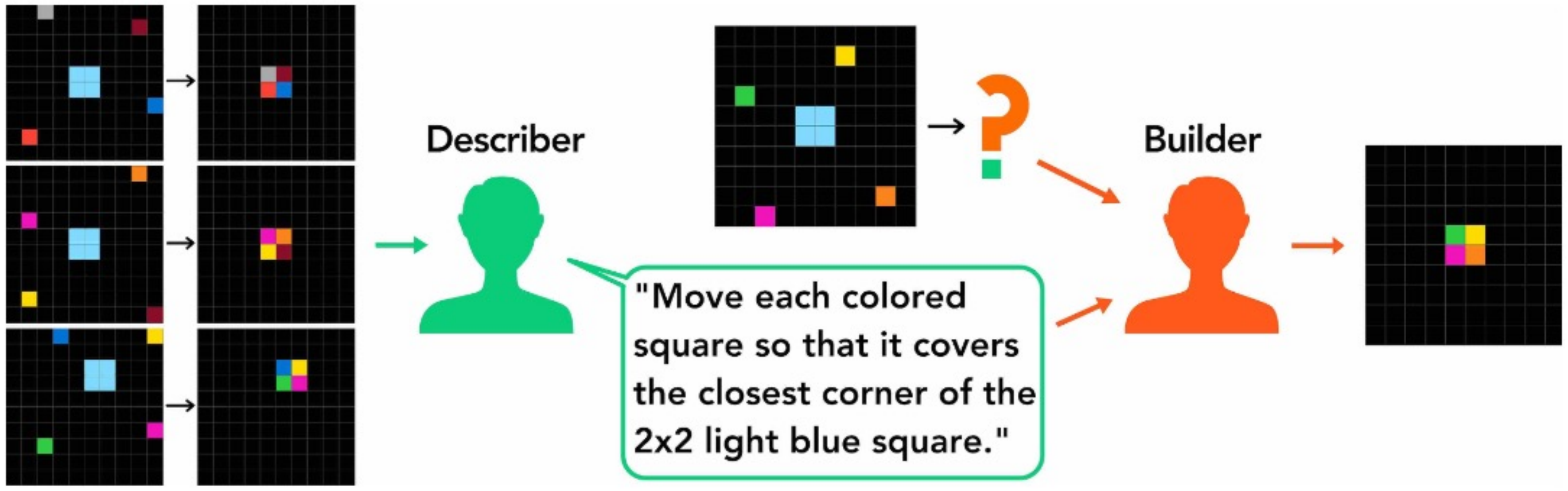
# Flexible Prompt Conditioning

- <Context>
- <Prompt>

- E.g.
- You are a salesperson who is courteous and firm in closing the deal.
- Here are the products and their costs: Apple ($3000), Banana ($2000), Carrot ($1000)
- Respond to the following potential customer message.
- “Your Apple product is too expensive! Give me cheaper options!”

## Language ARC

- 88% of the original ARC tasks can be represented in a text instruction where another human can solve it without needing the input-output examples


Describer
"Move each colored square so that it covers the closest corner of the 2x2 light blue square."
Builder

https://github.com/samacqua/LARC

## Initial Ideas

*An Approach to Solving the Abstraction and Reasoning Corpus (ARC) Challenge*

https://arxiv.org/abs/2306.03553

## Idea (V1)

- Get GPT to generate the language description of the task from input-output examples

- Use the language description to encode a python program to solve ARC

## Idea (V2)

- Get GPT to generate the language description of the task from input-output examples
  - Do broad approach, before generating specific approach

- Use the language description to apply on the json input to get a json output

## Idea (V3)

- Get GPT to generate the language description of the task from input-output examples
  - Do broad approach, before generating specific approach
  - Imbue human priors
  - Define standard actions model can take

- Use the language description to apply on the json input to get a json output

## Prompt (Part 1)

- You are given a series of inputs and output pairs.
- These are all in the form of a 2D array, representing a 2D grid, with values from 0-9.
- The values are not representative of any ordinal ranking.
- Input/output pairs may not reflect all possibilities, you are to infer the simplest possible relation making use of symmetry and invariance as much as possible.

- The input can be something like:
  - entire grid being the sandbox to manipulate
  - using a part of the grid (individual squares or portions of the grid) to depict instructions of how to do the task.
  - Position and symmetry is very important. > using regions of similar value to depict area for answer of the task

- The output can be something like:
  - same output size as input after performing action
  - output one of the fixed predetermined patterns used to classify the input image
  - using output to show the ordering of objects, such as by size, height, width, position, value

## Prompt (Part 2)

- Each of the input-output relation can be done with one or more actions chained together, which could be something like (not exhaustive):
- object view (defined as continuous squares connected horizontally, vertically and/or diagonally, separated by 0 values)
  - objects can be the of the same value, or different values combined together
  - objects may be hidden beneath other objects
  - rotating or shifting objects
- pixel view
  - rotation / reflection symmetry
  - continuation of a pattern
  - changing values
- segment view
  - combine two segments of the input into one single one based on a simple rule
  - rule can be certain values are prioritized over others, or combination of values into new ones

- Do the following:
  - What is the broad description of the input/output relation that holds for all input/output pairs?
  - What is the step by step description of the input/output relation that holds for all input/output pairs?
  - Apply this description to the test input and find out the answer 'to_be_filled'.

# Example

Here is the output for Public Evaluation Task 162 (68b67ca3.json) using GPT4:

*Broad description of the input/output relation:*
*The input/output relation involves extracting non-zero elements from specific positions in the input grid and arranging them into a smaller 3x3 output grid, maintaining their relative positions.*
*Step by step description of the input/output relation:*
*Identify non-zero elements in the following positions of the input grid: (1,1), (1,3), (1,5), (3,1), (3,3), (3,5), (5,1), (5,3), and (5,5). Place these non-zero elements into the corresponding positions in a 3x3 output grid. Applying this description to the test input:*
*Test input: [[3, 0, 3, 0, 4, 0], [0, 0, 0, 0, 0, 0], [7, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0], [7, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0]]*
*Following the step-by-step description, we extract the non-zero elements from the specified positions:*
*(1,1): 3, (1,3): 3, (1,5): 4, (3,1): 7, (3,3): 0, (3,5): 1, (5,1): 7, (5,3): 0, (5,5): 1*
*Place these elements into the 3x3 output grid: [[3, 3, 4], [7, 0, 1], [7, 0, 1]]*
*So, the output for the test input is: [[3, 3, 4], [7, 0, 1], [7, 0, 1]]*

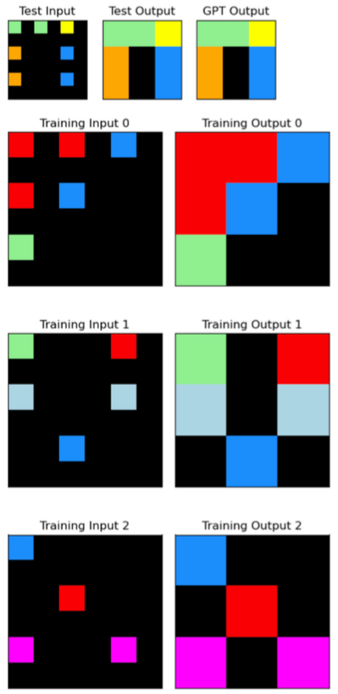


Figure 5: The Test Output and GPT Output for Public Evaluation Task 162 (68b67ca3.json)

## Limitations

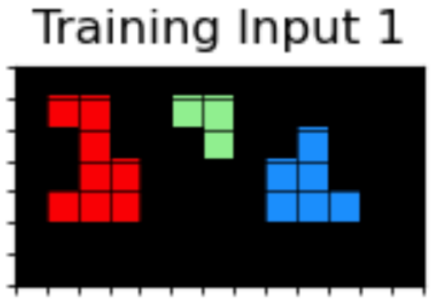
- Limited context length to store json representations of large grids / multiple input-output samples and to store instructions
- Limited understanding of what an object is from the json file
- Limited fact-checking abilities to determine if the input-output relation derived is correct

## Improvements I have made

## Solution: Limited Context Length

- Convert Input/Output to List of Strings Format
- Use letters to avoid bias of adding/subtracting/modulo pixel values

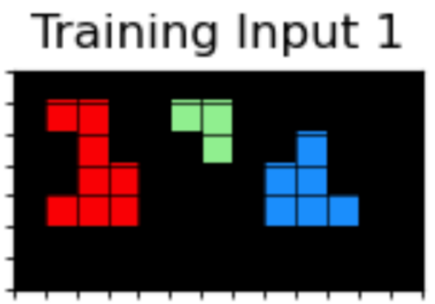
- `'input':`
- `[`
- `'.............',`
- `'.bb..cc......',`
- `'..b...c..a...',`
- `'..bb....aa...',`
- `'.bbb....aaa..',`
- `'.............',`
- `'.............'`
- `]`



## Solution: Object View

- GPT is not that good at viewing objects
- Imbue it a view to do so

```
'input_objects':
[{'tl': (1,1),'shape': ['xx.','.x.','.xx','xxx'],'val': 'b'},
{'tl': (1,5),'shape': ['xx','.x'],'val': 'c'},
{'tl': (2,8),'shape': ['.x.','xx.','xxx'],'val': 'a'}]
```



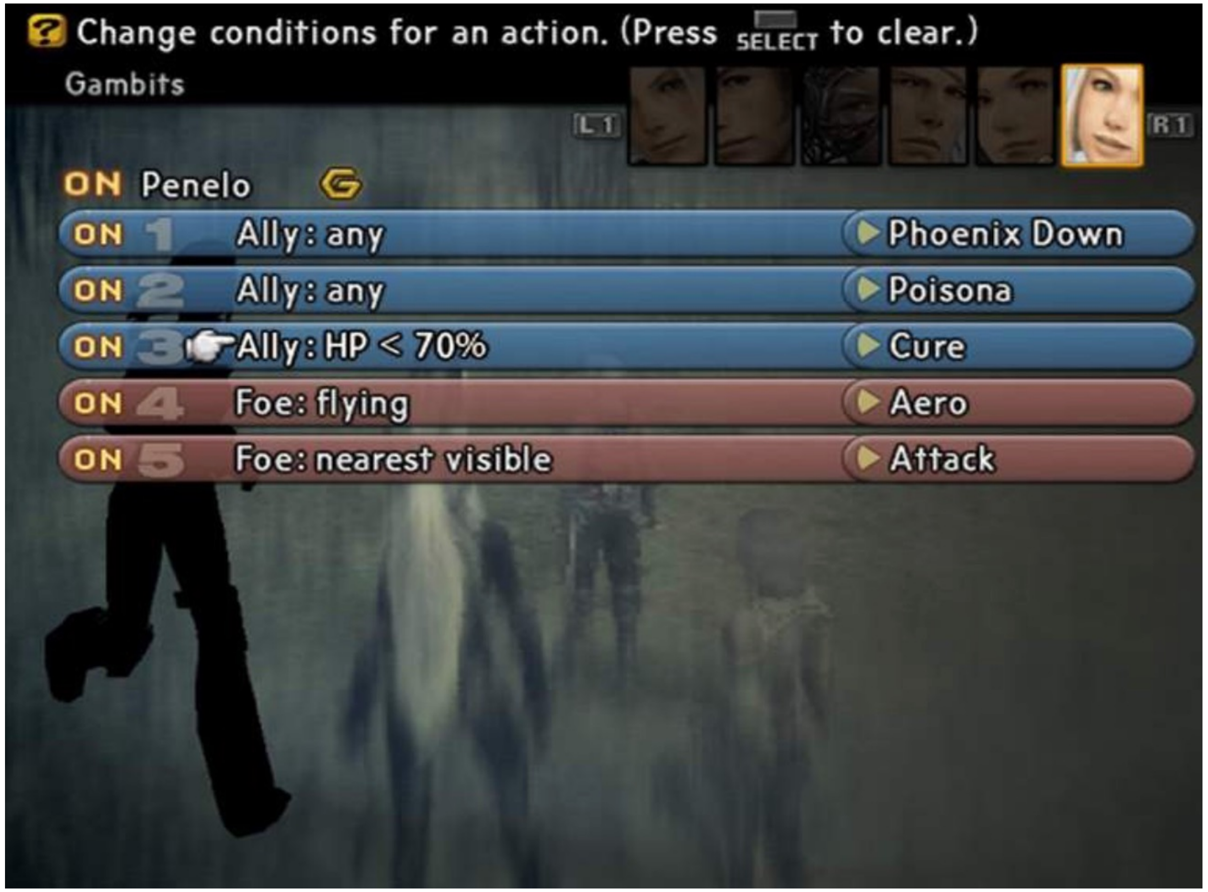
# GPT is bad at fact-checking

- Instead of language description, use Python Code instead
- Text-based biases is not enough - Give some helper functions to ground GPT generation

- Example Functions
  - - **get_objects(grid, diag=False, empty=False):** Takes in grid, diag (can be set to True if we want diagonals to be considered as same object) and empty (set to True if we want empty cells to form objects too), returns object dictionary comprising top-left coordinate of object ('tl'), shape of object in list of strings format ('shape'), value of object ('val')]
    - I ['.aaa', '..ba']
    - O [{'tl': (0, 1), 'shape': ['xxx', '..x'], 'val': 'a'}, {'tl': (1, 2), 'shape': ['x'], 'val': 'b'}]
  - - **fill_object(grid, obj_dict):** returns grid filled with the given object
    - I ['....','....'], {'tl': (0, 1), shape': ['xxx', '..x'],'val': 'a'})
    - O ['.aaa', '...a’]
  - - **fill_coord(grid, coord, value):** fills up grid at given coord with value
    - I ['...','...'], (1,1), 'a')
    - O ['...', '.a.']
  - - **fill_line_between_coords(grid, coord1, coord2, value):** fills up a line from coord1 to coord2 with value
    - I ['...','...'], (0, 0), (1, 1), 'a'
    - O ['a..', '.a.']

## Imbuing Flexibility: Conditional Code

- Condition followed by function/action
- Will not have runtime errors
- Can potentially be hierarchical if it calls another function as the outcome


Change conditions for an action. (Press SELECT to clear.)

Gambits

ON Penelo

| | | | |
|---|---|---|---|
| ON | 1 | Ally: any | Phoenix Down |
| ON | 2 | Ally: any | Poisona |
| ON | 3 | Ally: HP < 70% | Cure |
| ON | 4 | Foe: flying | Aero |
| ON | 5 | Foe: nearest visible | Attack |

Final Fantasy XII, Gambit System

# Conditional Code

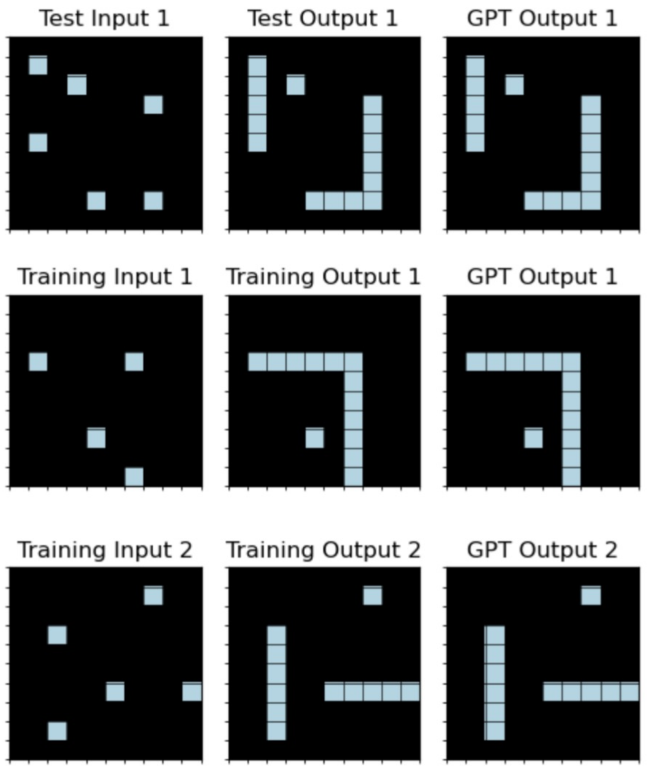
```
Each helper function can be conditional. The conditions can be:
- by attribute, such as cell_num, shape, color, size of object, maximum, minimum
- the condition can be an attribute on all objects, for instance, objects with
the most common colour, or objects with the most common shape
- by position, such as if neighbouring cell is of a certain type or shape

Conditions (non-exhaustive):
- on_same_line(coord1, coord2, line_type): Returns True/False if coord1 is on the same line as coord2.
line_type can be one of ['row', 'col', 'diag']
(1, 1), (2, 2), 'diag'
@ True

The overall relation should be encoded as such:
if {condition}: {helper function}
```

```
def transform_grid(grid):
        objects = get_objects(grid)
        new_grid = [list(row) for row in grid]
        for obj1 in objects:
                for obj2 in objects:
                        if on_same_line(obj1['tl'], obj2['tl'], 'row'):
                                new_grid = fill_line_between_coords(new_grid, obj1['tl'],
obj2['tl'], obj1['value'])
                        if on_same_line(obj1['tl'], obj2['tl'], 'col'):
                                new_grid = fill_line_between_coords(new_grid, obj1['tl'],
obj2['tl'], obj1['value'])
        return [''.join(row) for row in new_grid]
```

ARC Training Set
ded97339.json



# More Efficient than Sequence of Unary Functions

```
def transform_grid(grid):
        objects = get_objects(grid)
        new_grid = [list(row) for row in grid]
        for obj1 in objects:
                for obj2 in objects:
                        if on_same_line(obj1['tl'], obj2['tl'], 'row'):
                                new_grid = fill_line_between_coords(new_grid, obj1['tl'],
obj2['tl'], obj1['value'])
                        if on_same_line(obj1['tl'], obj2['tl'], 'col'):
                                new_grid = fill_line_between_coords(new_grid, obj1['tl'],
obj2['tl'], obj1['value'])
        return [''.join(row) for row in new_grid]
```

Conditional Code

```
def solve_ded97339(I):
    x1 = ofcolor(I, EIGHT)
    x2 = product(x1, x1)
    x3 = fork(connect, first, last)
    x4 = apply(x3, x2)
    x5 = fork(either, vline, hline)
    x6 = mfilter(x4, x5)
    O = underfill(I, EIGHT, x6)
    return O
```

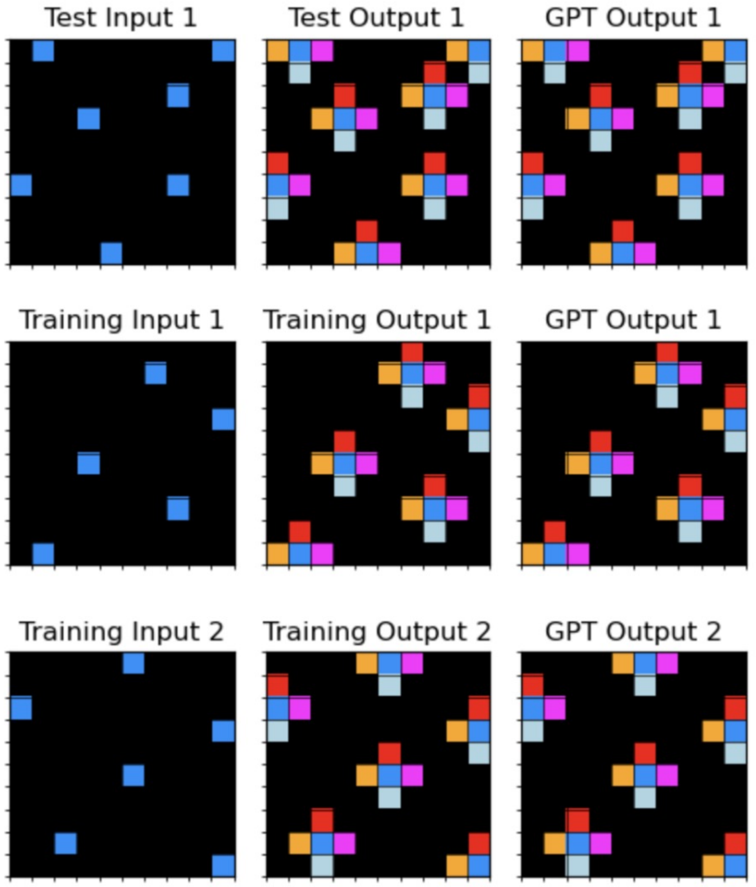
Michael Hodel's DSL
https://github.com/michaelhodel/arc-dsl

# Refined prompt to do Broad to Specific

- Do not provide explanations.

- Output only the following in .json format:
- **"reflection":** {reflection},
- **"description_pixel":** describe the grid in terms of pixels for input-output pairs, highlighting possible operations (e.g. rotation, reflection) on the entire grid or subsections of the grid
- **"description_object":** describe the grid in terms of objects for input-output pairs, highlighting the similarities and differences,
- **"description_mapping":** describe the mapping of input to output in terms of grid sections
- **"overall_pattern":** describe the simplest input-output relationship for all input-output pairs,
- **"program_instructions":** specific instructions to encode overall_pattern into a python function,
- **"python_program":** python function named 'transform_grid' that takes in a list of strings (2D grid representation) and generates a list of strings (2D grid representation). You may use helper functions. Output as a string in a single line with \\n and \\t.

# Example: ARC Training Set d364b489.json



# Final GPT Output

- **'reflection':** "There is no change in grid size between the input and output pairs, they all remain the same size. The number of objects increases from input to output, with multiple new pixels appearing around the original 'a' pixels. The original 'a' pixels do not move, but new pixels are introduced around them. The object sizes generally increase, with single pixel 'a' objects in the input becoming larger multi-pixel objects in the output.",

- **'description_pixel':** "There doesn't appear to be any consistent whole-grid transformations such as rotation or reflection. However, a common operation involves placing pixels around the original 'a' pixel in the grid. In particular, pixels with values 'g', 'a', 'f', 'h', and 'b' are placed around the original 'a' pixel following a specific pattern. This operation happens independently for each 'a' pixel in the grid.",

- **'description_object':** "The objects in the input are all single-pixel 'a' objects. In the output, each 'a' object is surrounded by other pixels forming a larger object. Specifically, an 'a' pixel is surrounded in an arc by 'g', 'f', 'h', and 'b' pixels in a clockwise direction starting from the left.",

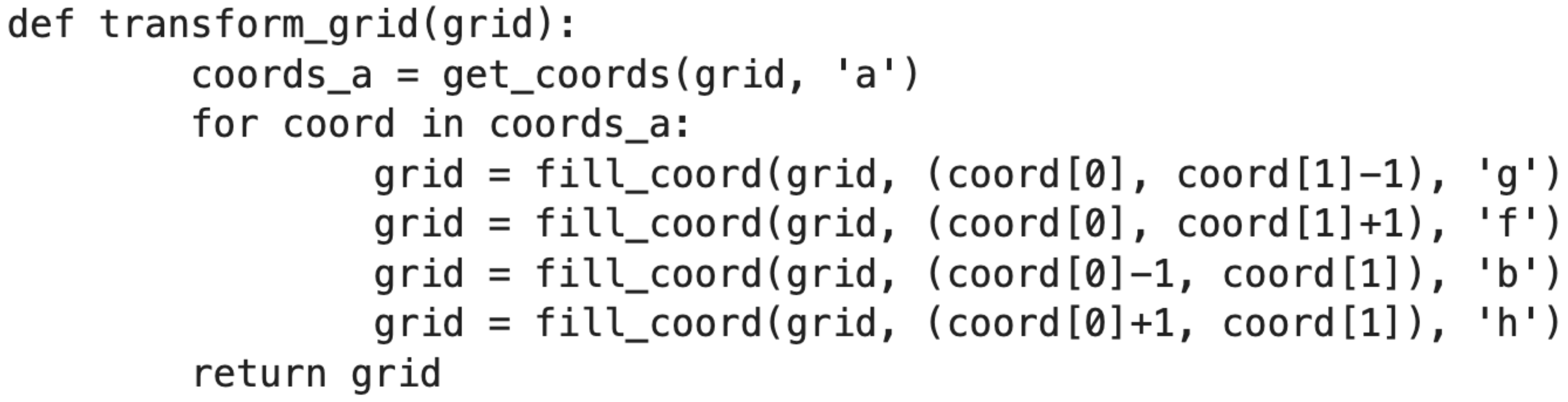
- **'description_mapping':** "The mapping doesn't involve transforming whole sections of the grid, but rather operates on individual 'a' pixels. Around each 'a' pixel, other pixels are placed following a specific pattern. This operation does not depend on the location of the 'a' pixel or the presence of other 'a' pixels in the grid.",

# Final GPT Output Part 2

- **'overall_pattern':** "For each 'a' pixel in the grid, place a 'g' pixel to the left, a 'f' pixel to the right, a 'h' pixel below, and a 'b' pixel above. This operation is applied independently for each 'a' pixel.",

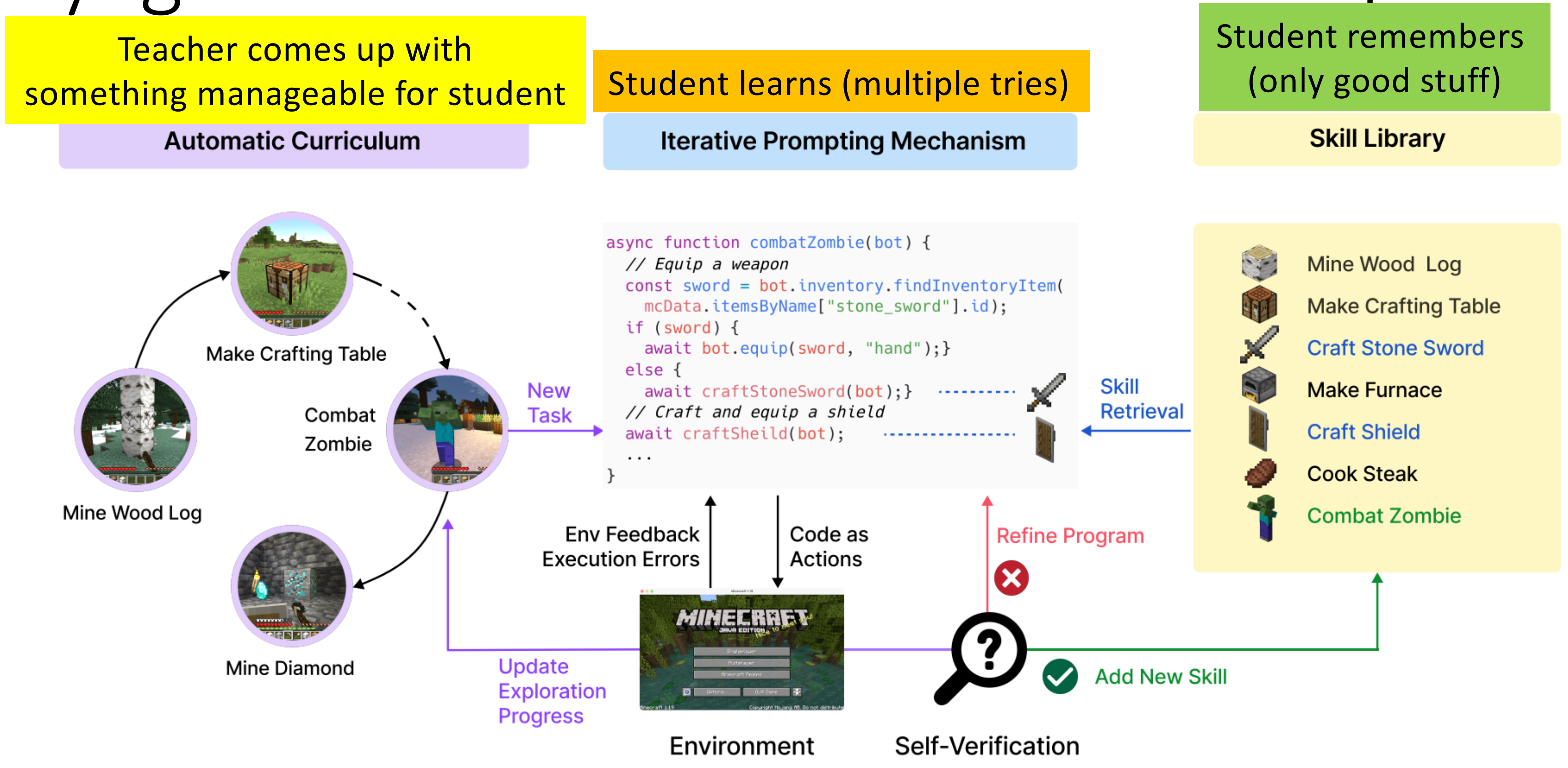
- **'program_instructions':** "1. Start with the original grid. 2. For each 'a' pixel in the grid, get its coordinates. 3. Place a 'g' pixel to the left of the 'a' pixel, a 'f' pixel to the right, a 'h' pixel below, and a 'b' pixel above. 4. Return the modified grid.",

- **'python_program':** "def transform_grid(grid):\n\tcoords_a = get_coords(grid, 'a')\n\tfor coord in coords_a:\n\t\tgrid = fill_coord(grid, (coord[0], coord[1]-1), 'g')\n\t\tgrid = fill_coord(grid, (coord[0], coord[1]+1), 'f')\n\t\tgrid = fill_coord(grid, (coord[0]-1, coord[1]), 'b')\n\t\tgrid = fill_coord(grid, (coord[0]+1, coord[1]), 'h')\n\treturn grid"



```
def transform_grid(grid):
        coords_a = get_coords(grid, 'a')
        for coord in coords_a:
                grid = fill_coord(grid, (coord[0], coord[1]-1), 'g')
                grid = fill_coord(grid, (coord[0], coord[1]+1), 'f')
                grid = fill_coord(grid, (coord[0]-1, coord[1]), 'b')
                grid = fill_coord(grid, (coord[0]+1, coord[1]), 'h')
        return grid
```

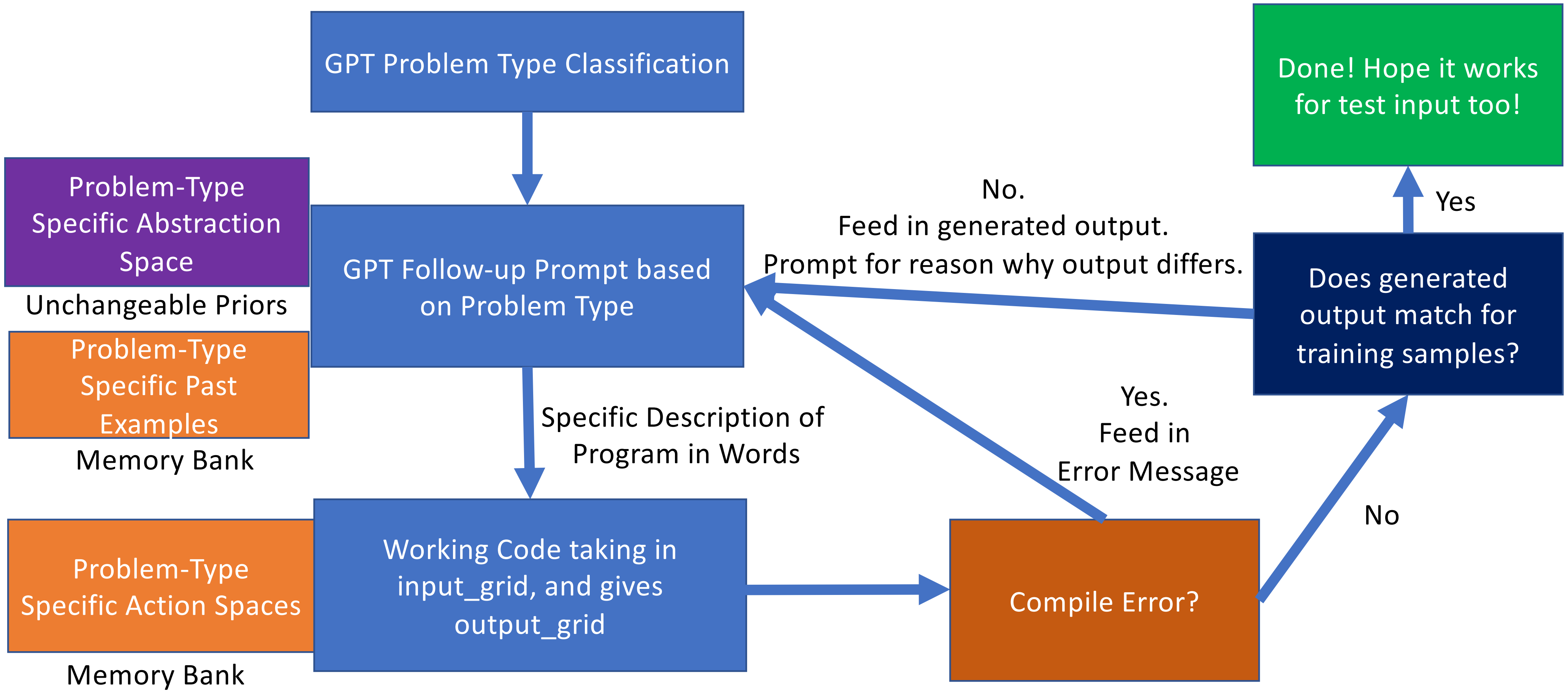
# Voyager – Recursive Environment Loop



```
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
  await craftSheild(bot);
  ...
}
```

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.

# Overall Flow of Generation

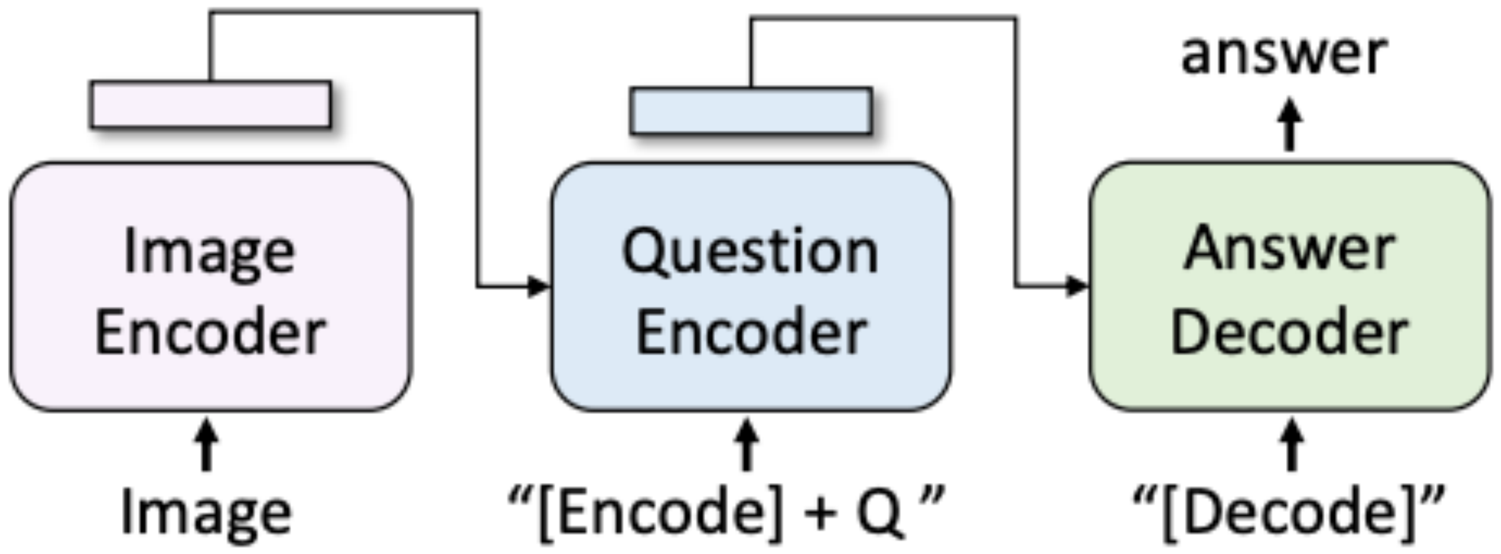

GPT Problem Type Classification
Problem-Type Specific Abstraction Space
Unchangeable Priors
GPT Follow-up Prompt based on Problem Type
Problem-Type Specific Past Examples
Memory Bank
Specific Description of Program in Words
Problem-Type Specific Action Spaces
Memory Bank
Working Code taking in input_grid, and gives output_grid
Compile Error?
Yes. Feed in Error Message
No
Does generated output match for training samples?
No. Feed in generated output. Prompt for reason why output differs.
Yes
Done! Hope it works for test input too!

## Live Demo

## Further Improvement Ideas

## Give better input representation

- Have a Visual Question-Answer module like BLIP
- Can ask multiple questions about the image
- BLIP not well tuned for ARC!

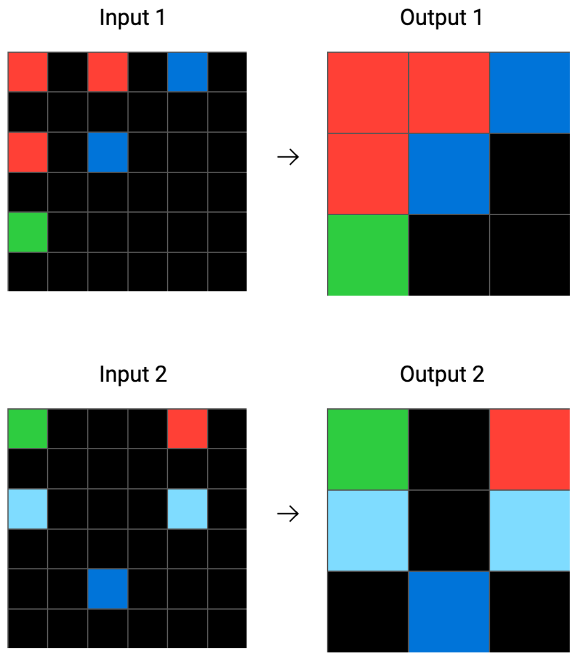


# Hierarchical Memory Referencing in Text

- **Broad Intent:** Reduce the input grid to a smaller size
  - Can reference/recite similar broad intents from memory to refine broad intent

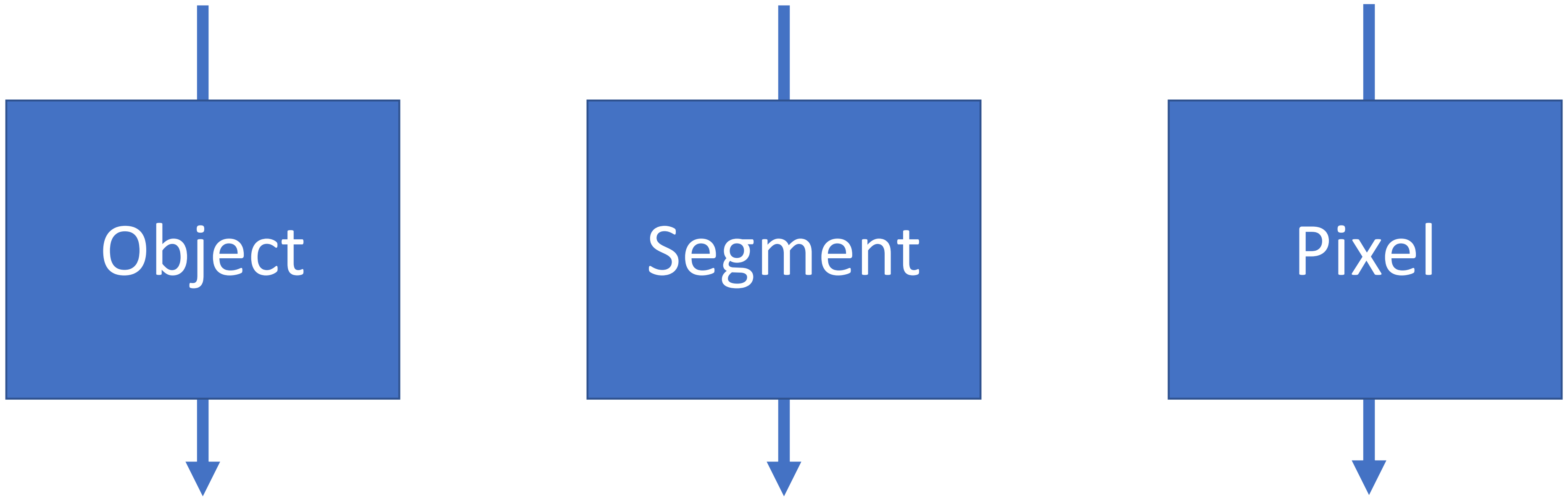
- **Detailed Steps (conditioned on Broad Intent):** Remove every other square from the row and columns of the grid
  - Can reference/recite similar detailed steps from memory to refine detailed steps

- Execution: Perform the detailed steps on the test input to get the answer



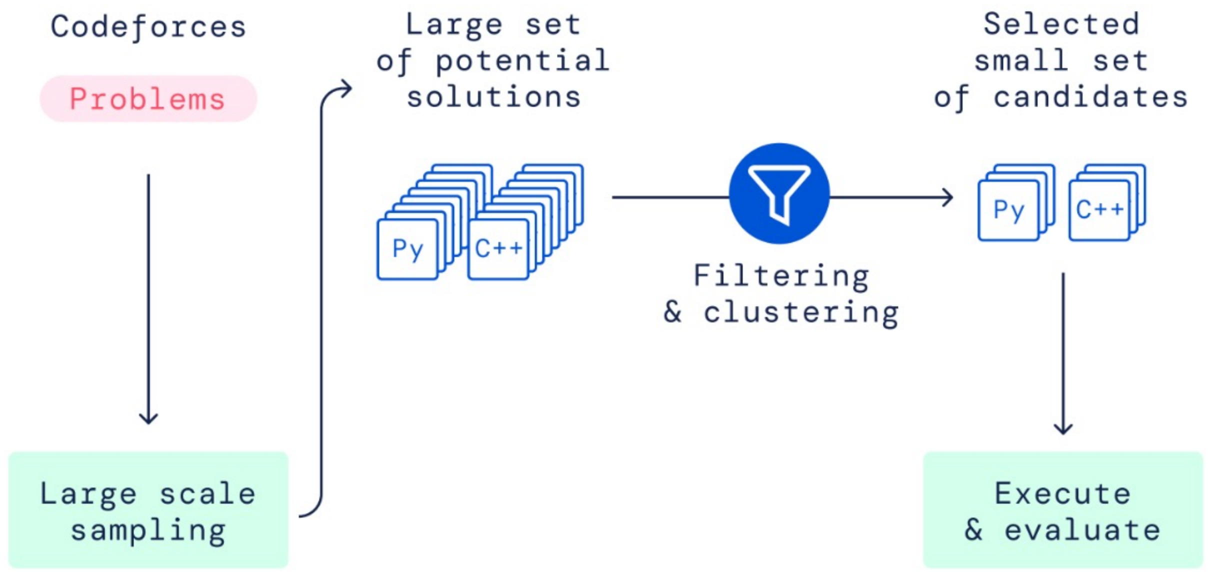
## Multiple Specialized Agents

- Each agent does certain actions at different hierarchies, like object, segment, or pixel level
- Generate multiple potential Python programs



# Generate massive solution set, then eliminate

- Evaluate each of the generated Python programs by various GPT agents
- Discard if fail to fit the training examples
- Choose the first performant one, or can also rank the “confidence” of the solution via prompting



AlphaCode

## Discussion