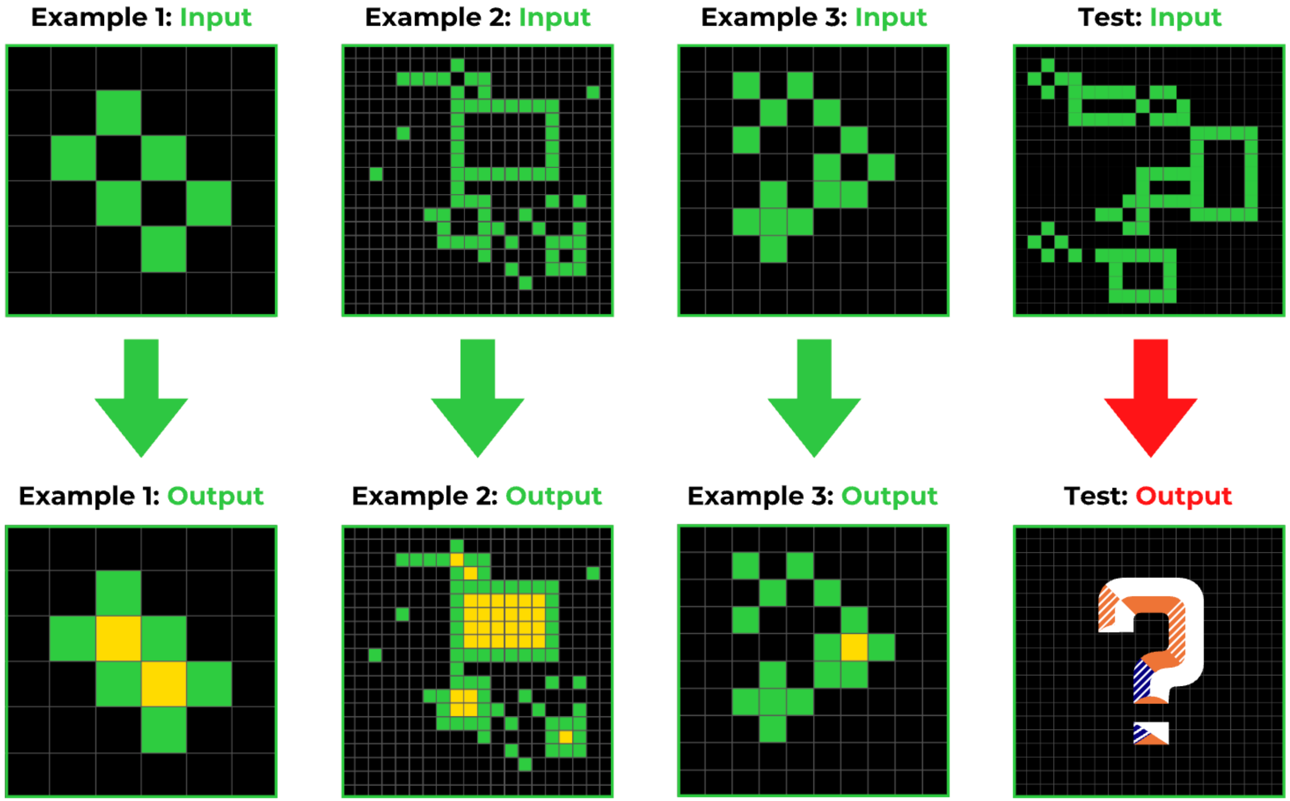
LLMs as a System to Solve the Abstraction and Reasoning Corpus (ARC) Challenge

John Tan Chong Min, Mehul Motani
National University of Singapore

Incorporating:

Multiple Abstract Spaces
Functional Prompting for Human Priors
Broad to Specific Prompting
Environment Feedback Loop

ARC Challenge



https://lab42.global/arc/

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

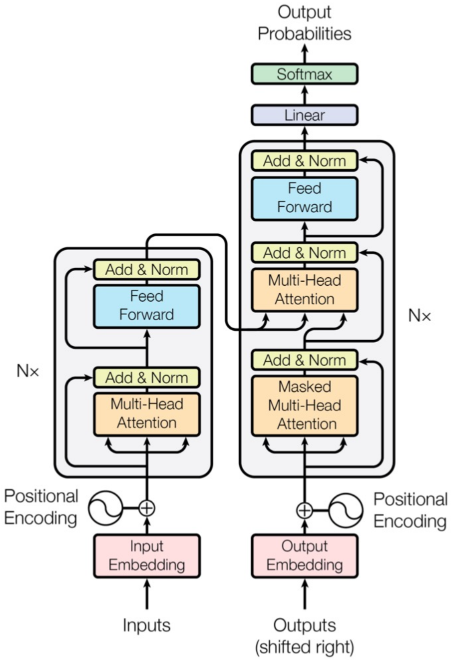


Figure 1: The Transformer - model architecture.

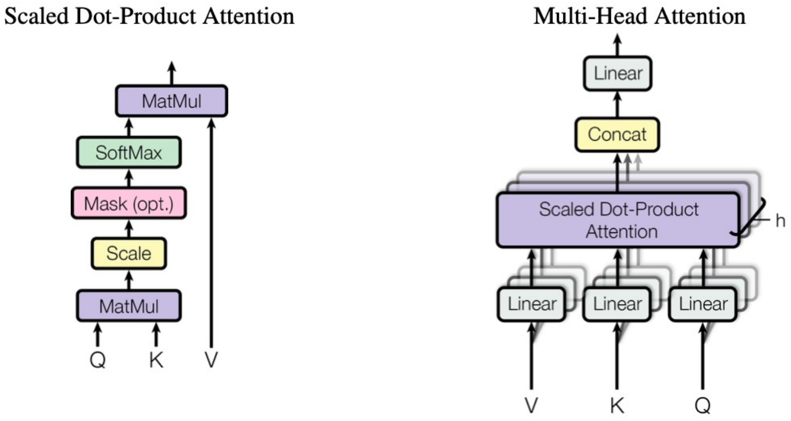


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

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

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

Transformers are versatile programs (1/2)

- Can be zero-shot prompted
  - You are a sentiment analysis tool …
- Can be few-shot prompted
  - These are a few examples: [Example A], [Example B], [Example C]
- Can be prompted sequentially
  - “Let’s think step by step” (Large Language Models are Zero-Shot Reasoners)
  - Broad to specific prompting to encourage better generation

Transformers are versatile programs (2/2)

- Can facilitate specific generation
  - Retrieval Augmented Generation with memory / relevant context

- Can use external tools to augment innate capabilities
  - Examples: Visual ChatGPT, OpenAI Functions

Transformers are not good at everything

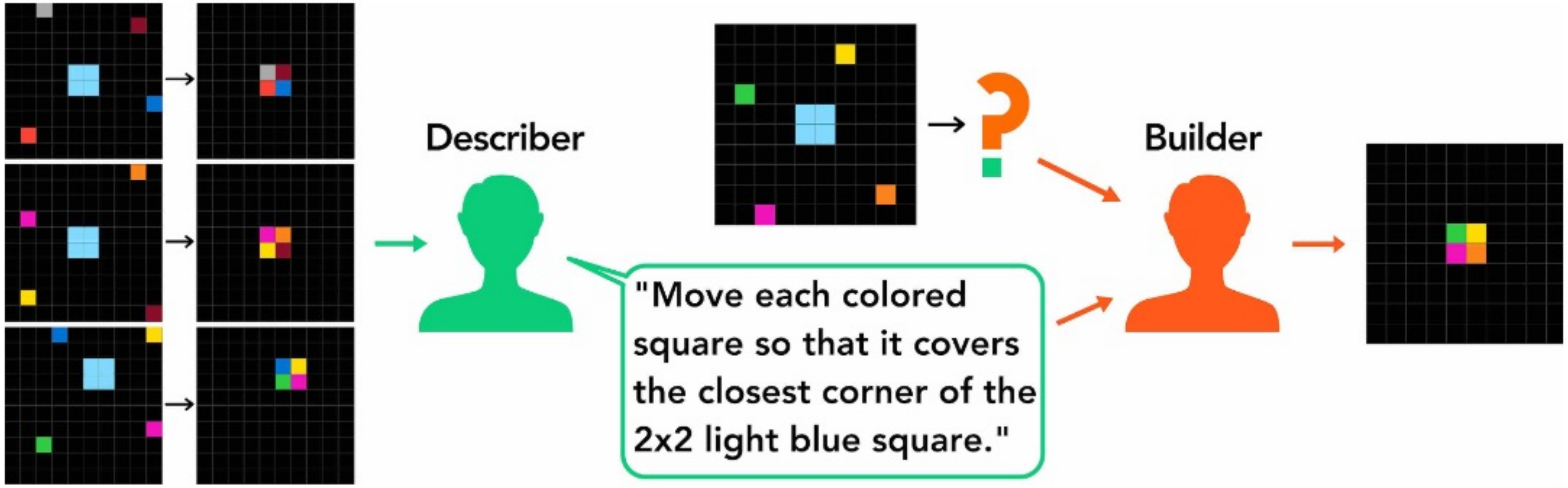
- Does not consistently follow rules
    - Solution: Use rule-based approaches if possible

- Does not know how to generate zero-shot functions of what is not commonly occurring in training set
    - Solution: Give some example functions to ground generation

- Next-token prediction is a 1D problem, ARC is a 2D problem
    - Convert 2D view into a single line of text for analysis
    - (Future - GPT-4V): Can the vision module help with representation?

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

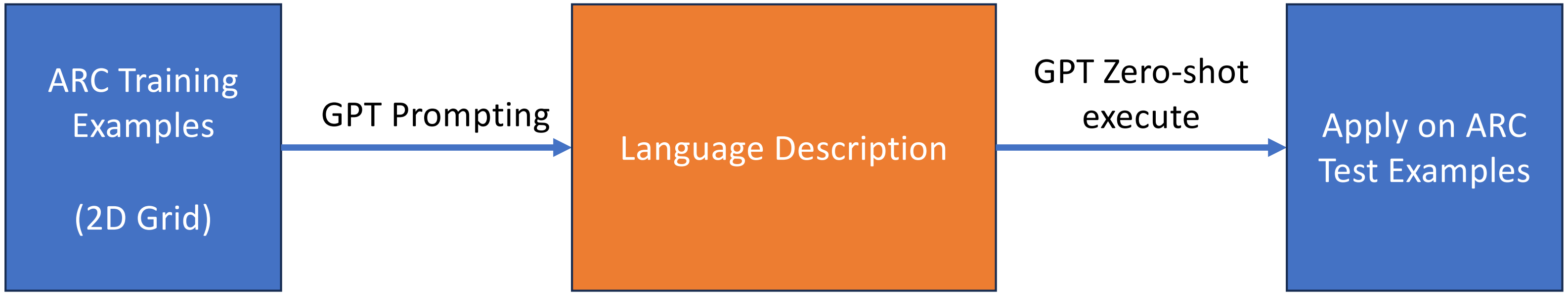

Describer
"Move each colored square so that it covers the closest corner of the 2x2 light blue square."
Builder

https://github.com/samacqua/LARC

The Seven Steps to ~~Greatness~~ Solving ARC

Or at least a general direction there…

Overall Process (v1)



ARC Training Examples

(2D Grid)

GPT Prompting

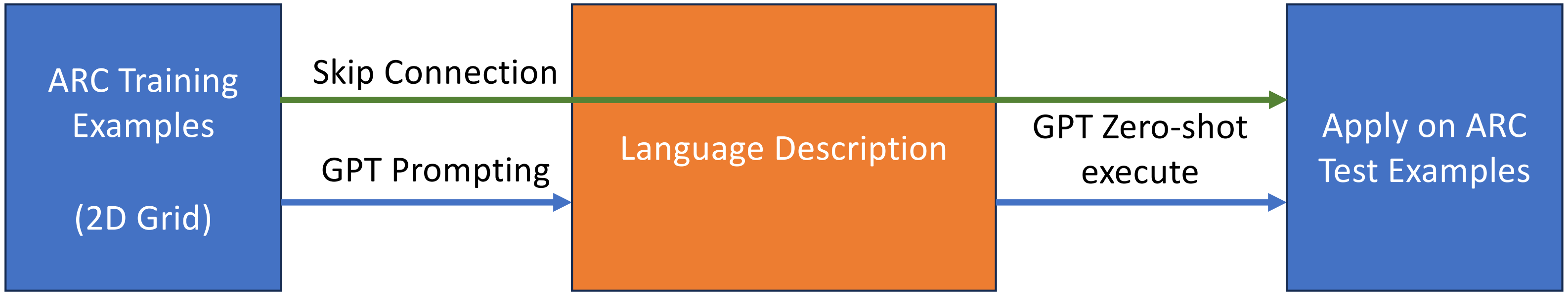
Language Description

GPT Zero-shot execute

Apply on ARC Test Examples

Problem: Language Description may compress too much information

Overall Process (v2)



ARC Training Examples

(2D Grid)

Skip Connection

GPT Prompting

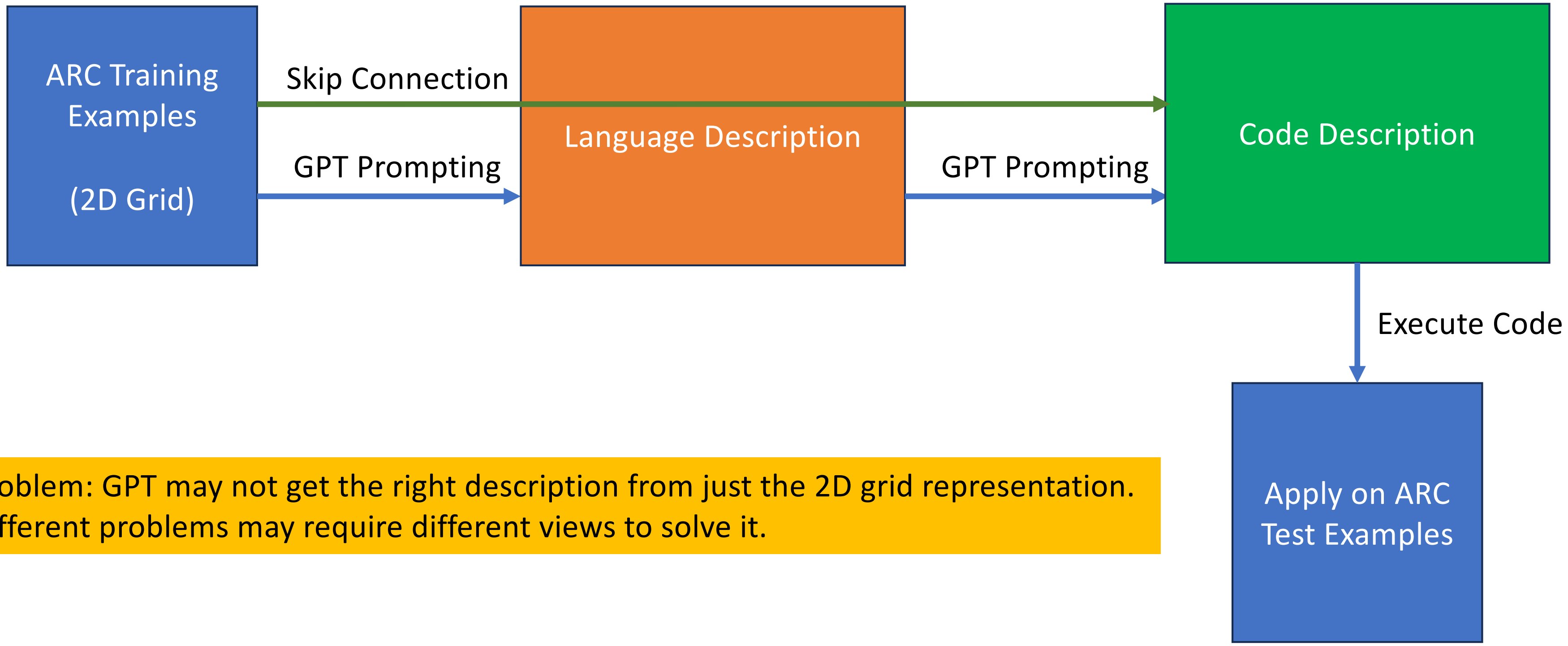
Language Description

GPT Zero-shot execute

Apply on ARC Test Examples

Problem: GPT is not reliable at executing instructions faithfully

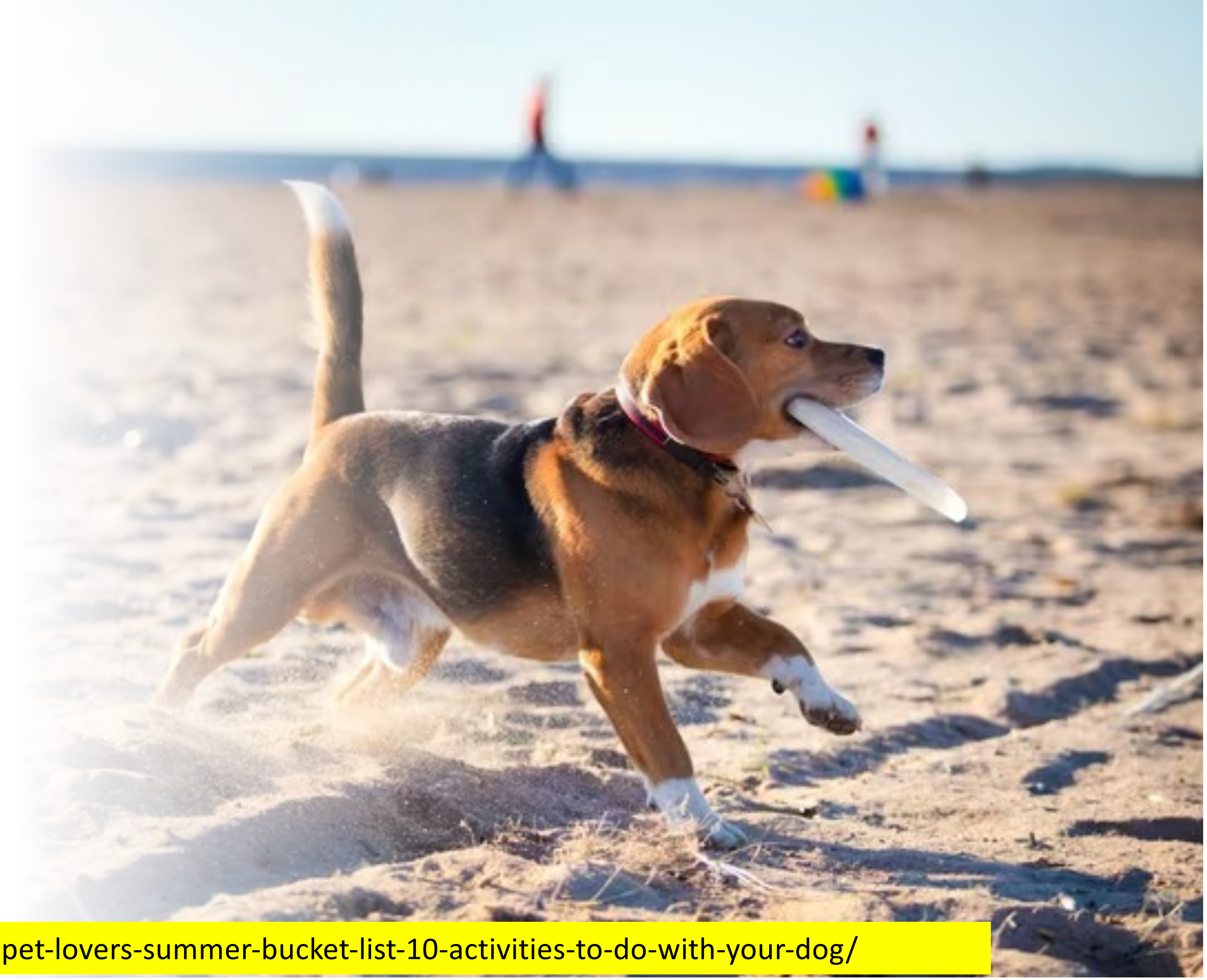
Overall Process (v3)



Problem: GPT may not get the right description from just the 2D grid representation. Different problems may require different views to solve it.

Image and Abstractions

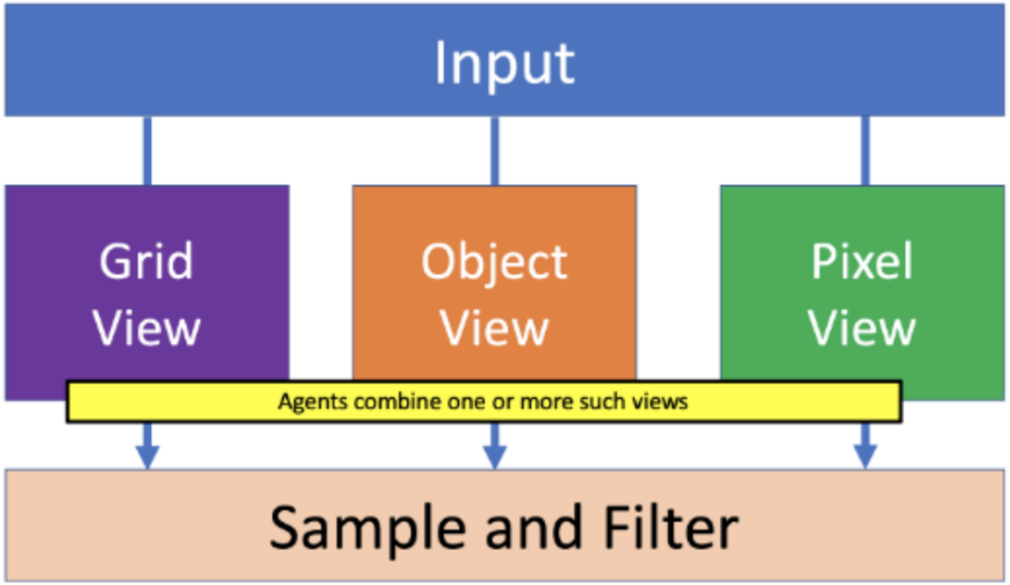
- One image
- Multiple views



https://be.chewy.com/pet-parenting-pet-lovers-summer-bucket-list-10-activities-to-do-with-your-dog/

Multiple Specialized Agents

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

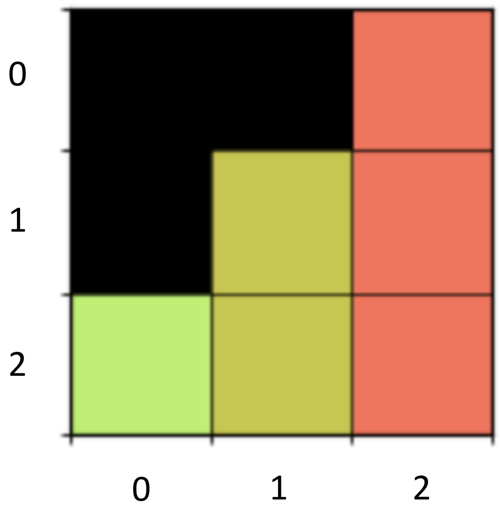


View Representations

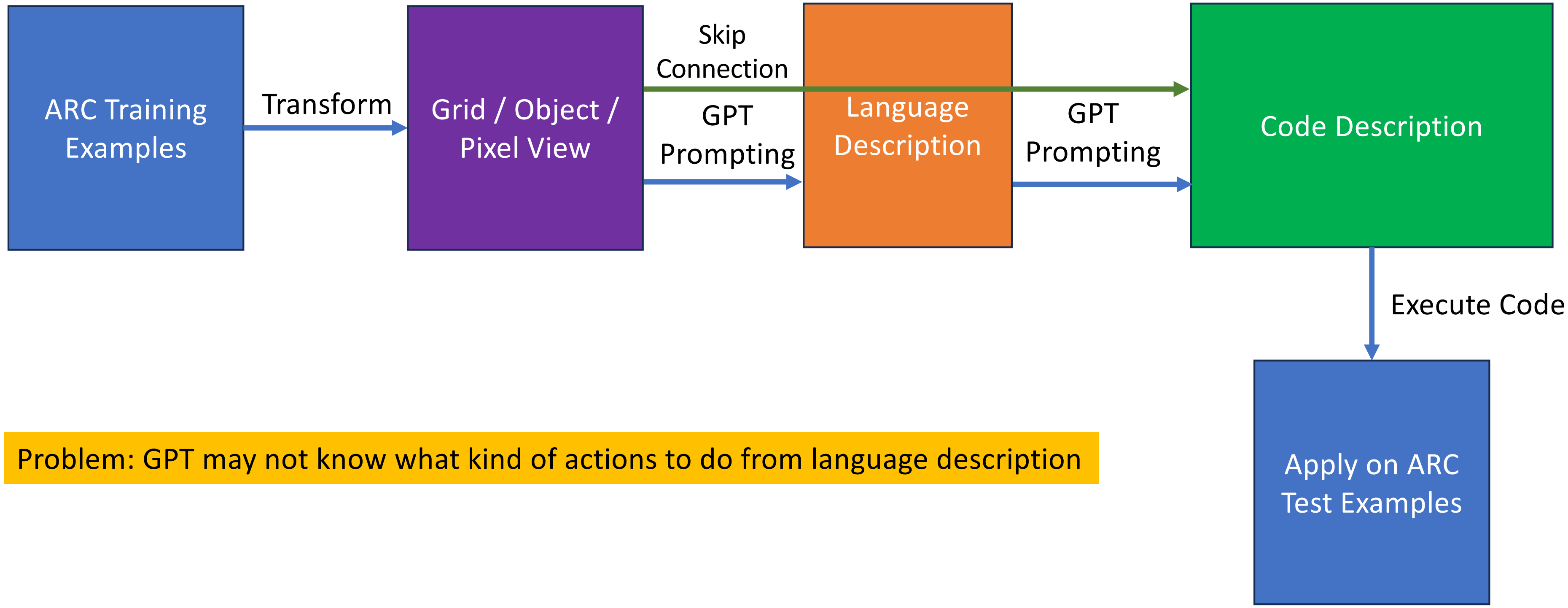
Grid View: [
[’.’, ’.’, ’f’],
[’.’, ’d’, ’f’],
[’c’, ’d’ ,’f’]]

Object View (Mono-Color):
[{’tl’:(0,2), ’grid’:[[’f’],[’f’],[’f’]], ’size’:(3,1), ’cell_count’:3, ’shape’:[[’x’],[’x’],[’x’]]},
{’tl’:(1,1), ’grid’:[[’d’],[’d’]], ’size’:(2,1), ’cell_count’:2, ’shape’:[[’x’],[’x’]]},
{’tl’:(2,0), ’grid’:[[’c’]], ’size’:(1,1), ’cell_count’:1, ’shape’:[[’x’]]}]

Pixel View: {
’f’:[(0,2),(1,2),(2,2)],
’d’:[(1,1),(2,1)],
’c’:[(2,0)]}



Overall Process (v4)



ARC Training Examples

Transform

Grid / Object / Pixel View

Skip Connection

GPT Prompting

Language Description

GPT Prompting

Code Description

Execute Code

Apply on ARC Test Examples

Problem: GPT may not know what kind of actions to do from language description

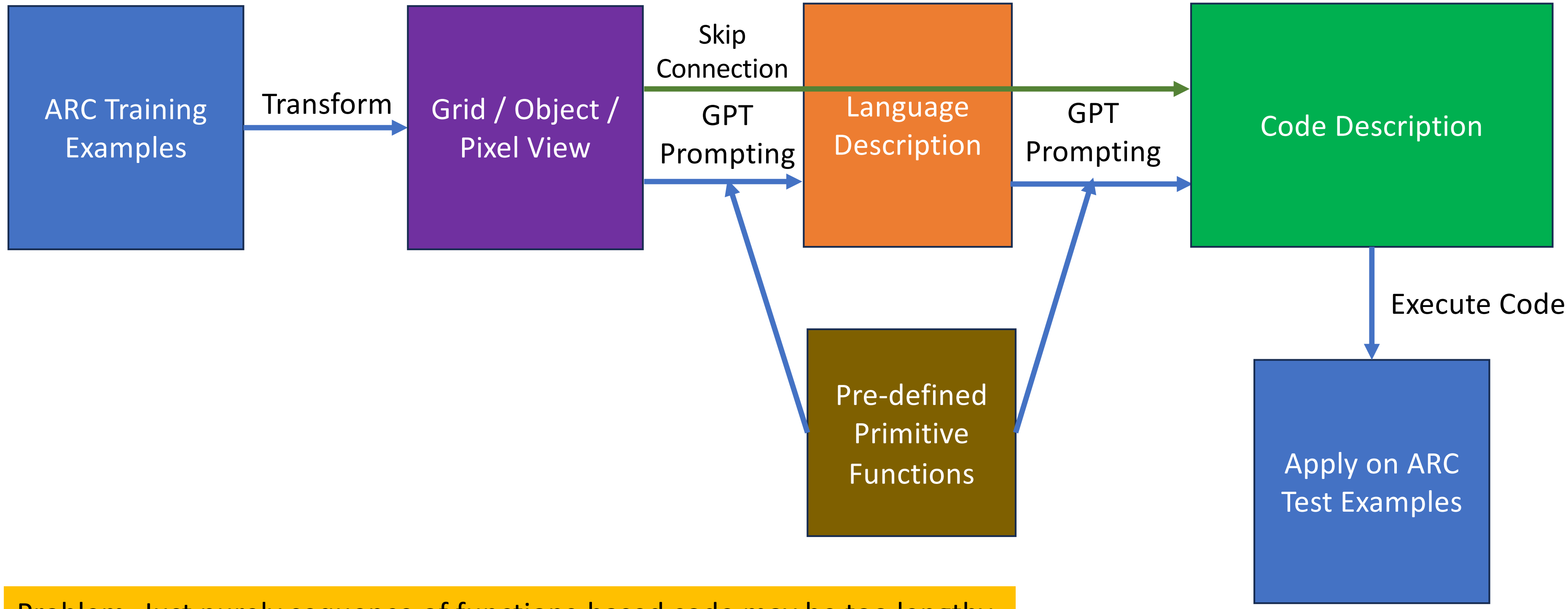
Primitive/Helper/Action Functions

- Add in some common functions that does the input-output relations

- Zero-shot Descriptions
    - horizontal_flip(grid): returns a horizontal flip of the grid
    - vertical_flip(grid): returns a vertical flip of the grid
    - replace(grid, grid_1, grid_2): replaces all occurences of grid_1 with grid_2 in grid

- One-shot Example
    - assert horizontal_flip([[’a’,’b’,’c’],[’d’,’e’,’f’]])==[[’c’,’b’,’a’], [’f’,’e’,’d’]]
    - assert vertical_flip([[’a’,’b’,’c’],[’d’,’e’,’f’]])==[[’d’,’e’,’f’],[’a’,’b’,’c’]]
    - assert replace([[’a’,’.’],[’a’,’a’]],[[’a’,’a’]],[[’c’,’c’]])==[[’a’,’.’],[’c’,’c’]]

Overall Process (v5)



Problem: Just purely sequence of functions-based code may be too lengthy

Conditional Code

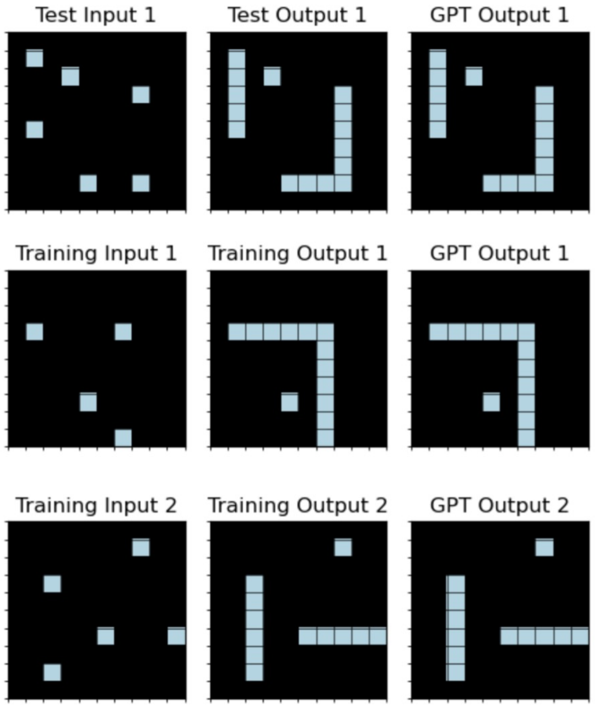
```
def transform_grid(grid):
        objects = get_objects(grid)
        new_grid = [list(row) for row in grid]
        for obj1 in objects:
                for obj2 in objects:
                        if on_same_line(obj1['tl'], obj2['tl'], 'row'):
                                new_grid = fill_line_between_coords(new_grid,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

ARC Training Set
ded97339.json



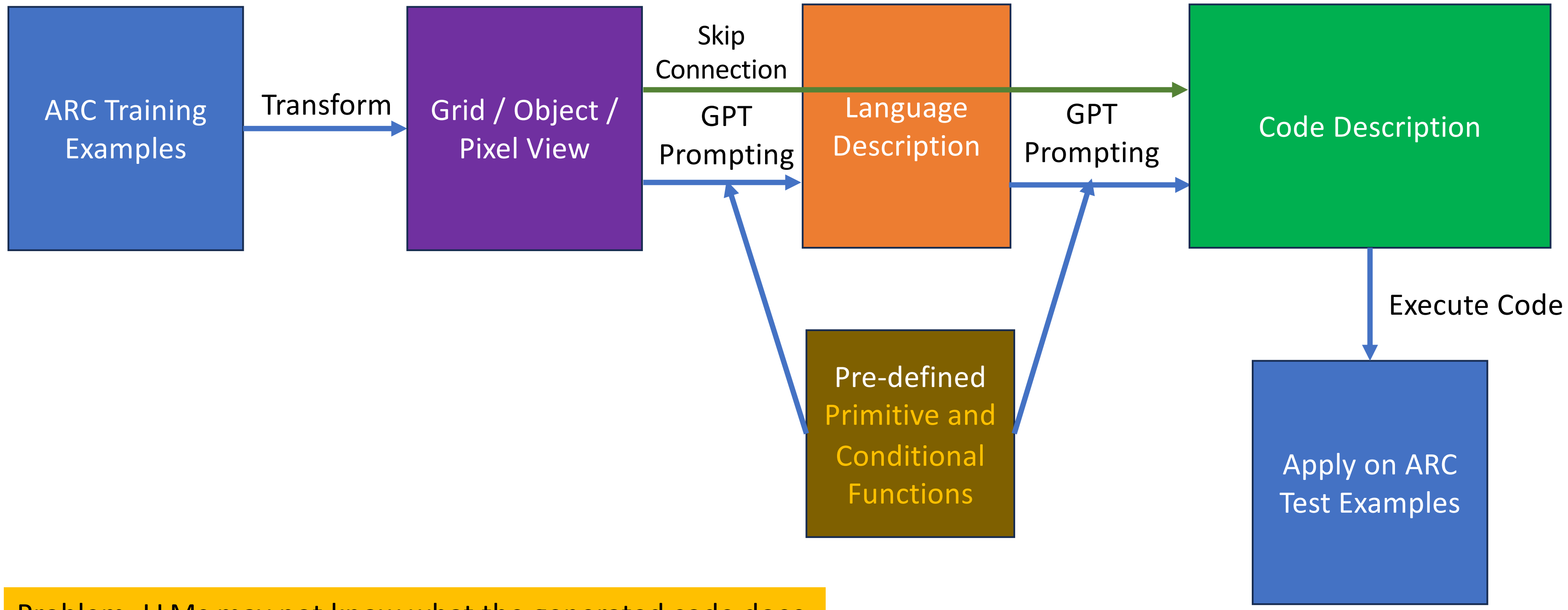
Conditional Functions

- Add in some common conditions for the input-output relation

- Zero-shot Descriptions
    - object_contains_color(obj, value): returns True/False if object contains a certain value
    - on_same_line(coord_1, coord_2): Returns True/False if coord_1 is on the same line as coord_2. line_type can be one of [’row’, ’col’, ’diag’]

- One-shot Example
    - assert object_contains_color({’tl’:(0,0),’grid’:[[’a’]]},’a’)==True
    - assert on_same_line((1,1),(1,2),’row’)==True
    - assert on_same_line((1,1),(2,1),’col’)==True
    - assert on_same_line((1,1),(2,2),’diag’)==True

Overall Process (v6)



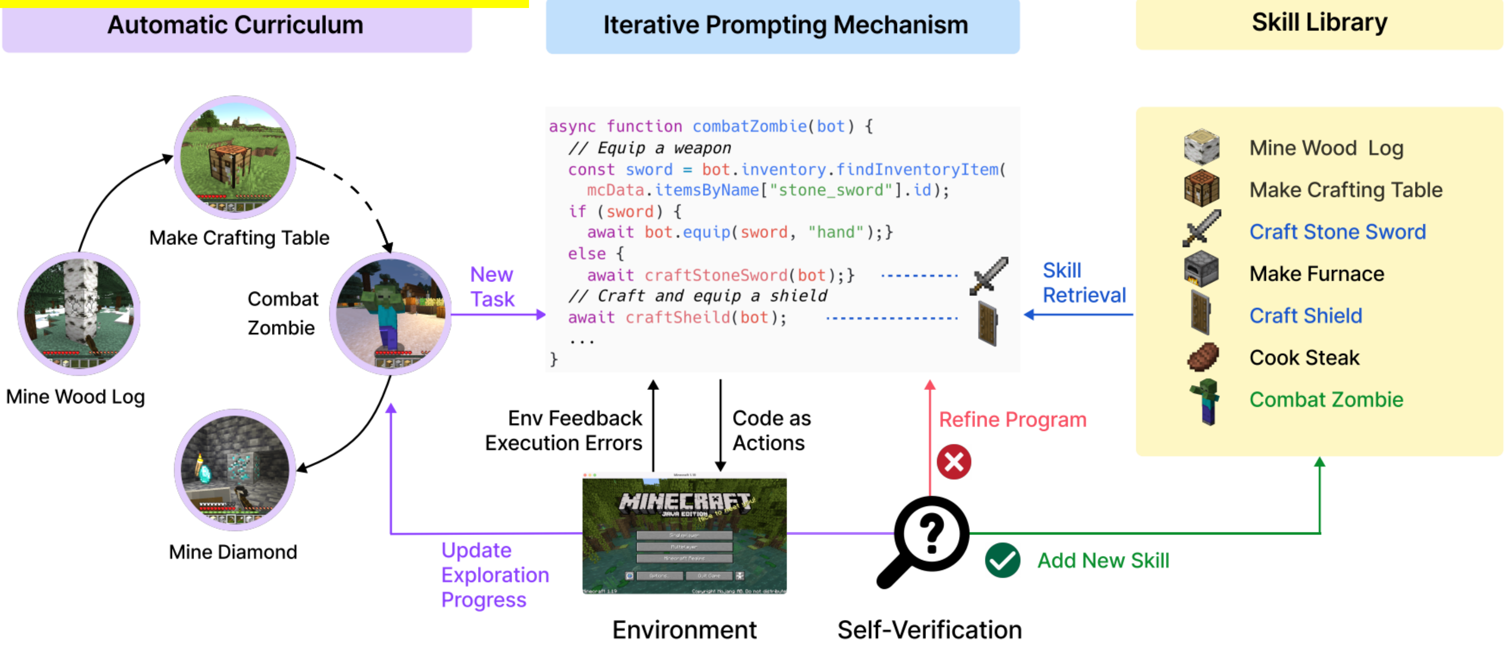
Problem: LLMs may not know what the generated code does

Voyager – Recursive Environment Loop

Teacher comes up with something manageable for student

Student learns (multiple tries)

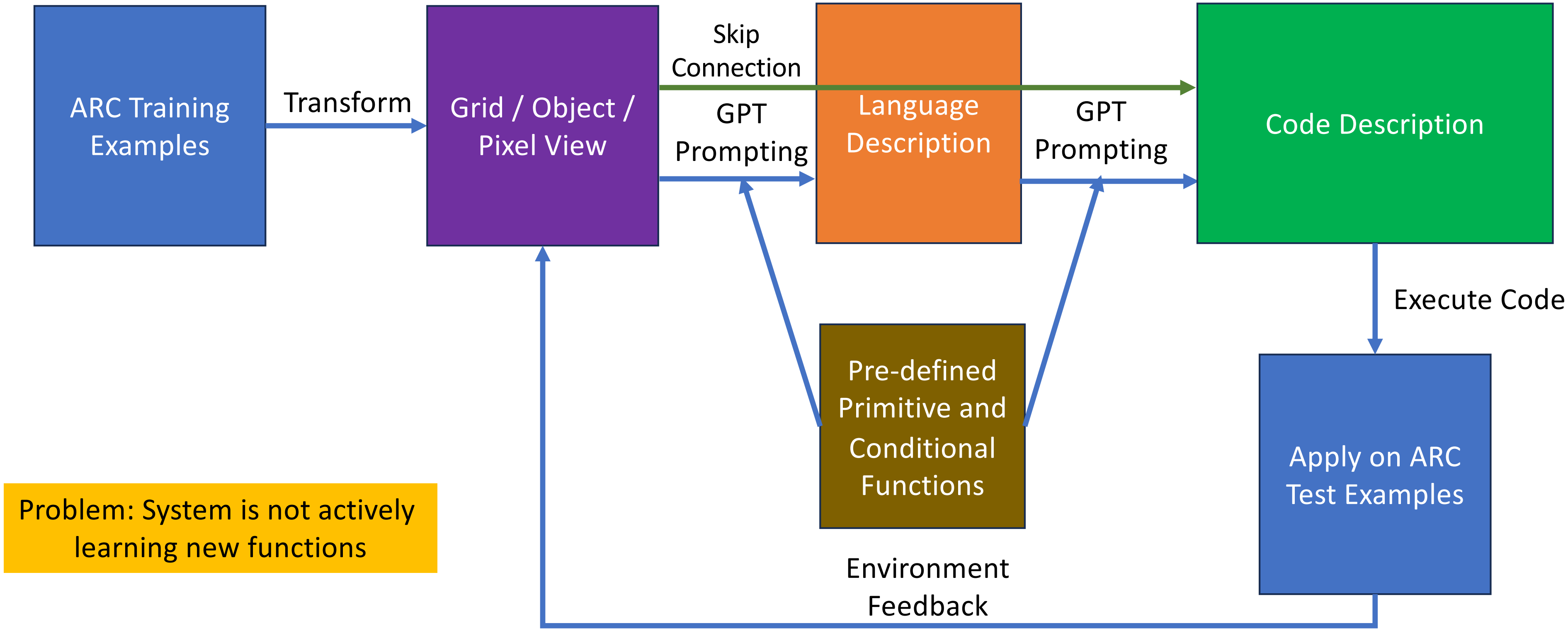
Student remembers (only good stuff)



```
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);
  ...
}
```

Voyager: An Open-Ended Embodied Agent with Large Language Models. Wang et al. 2023.

Overall Process (v7)



ARC Training Examples

Transform

Grid / Object / Pixel View

Skip Connection

GPT Prompting

Language Description

GPT Prompting

Code Description

Execute Code

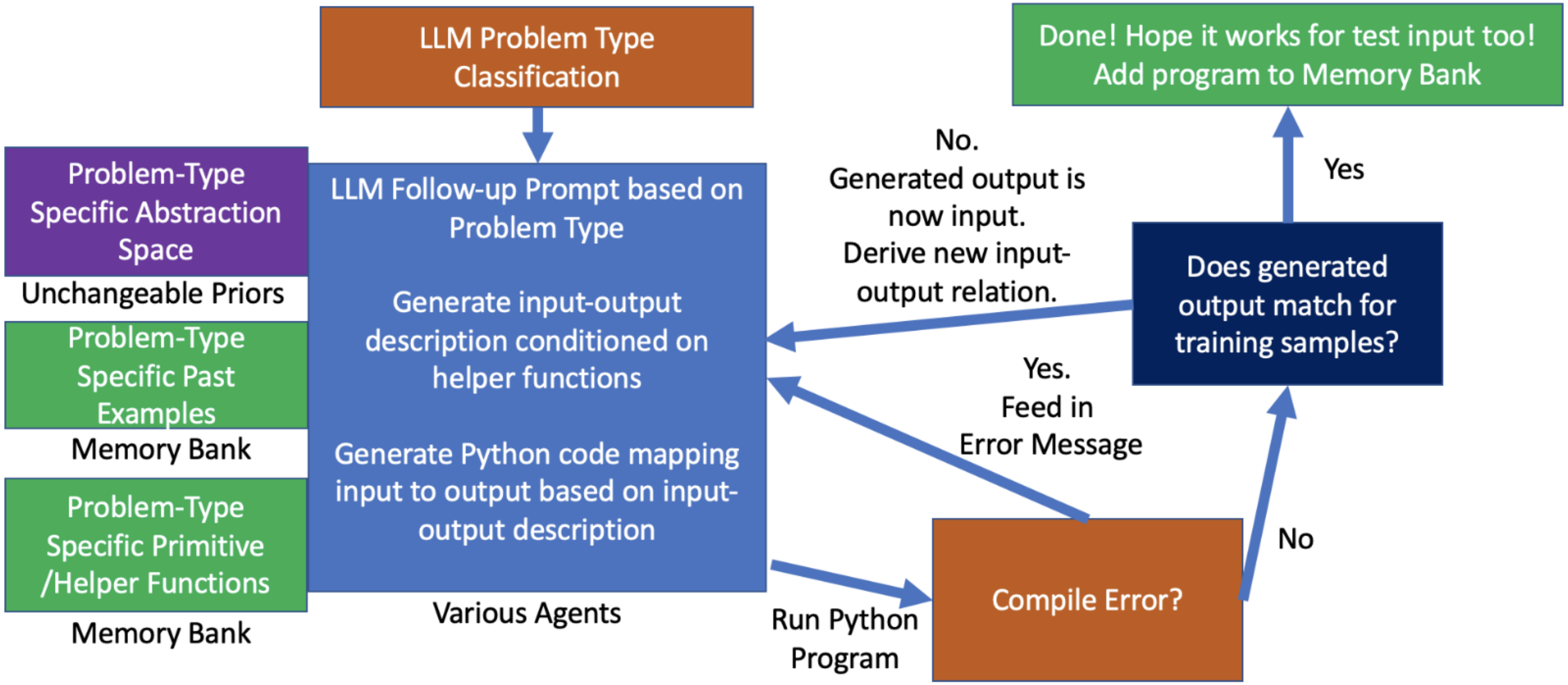
Pre-defined Primitive and Conditional Functions

Apply on ARC Test Examples

Environment Feedback

Problem: System is not actively learning new functions

Overall Flow of Generation with Memory



LLM Problem Type Classification

Problem-Type Specific Abstraction Space

Unchangeable Priors

Problem-Type Specific Past Examples

Memory Bank

Problem-Type Specific Primitive /Helper Functions

Memory Bank

LLM Follow-up Prompt based on Problem Type

Generate input-output description conditioned on helper functions

Generate Python code mapping input to output based on input-output description

Various Agents

Run Python Program

Compile Error?

Yes. Feed in Error Message

No

Does generated output match for training samples?

No. Generated output is now input. Derive new input-output relation.

Yes

Done! Hope it works for test input too! Add program to Memory Bank

Results

Evaluation Details

- 111 out of 400 Training Set Problems which fit into 3000 tokens using Grid View and Object View

- OpenAI GPT4 API temperature of 0.7 **or** GPT4 Web Browser Interface

- Views Used: Total 20
    - Grid View (default)
    - Object View (10 types – Color Type: Multicolor / Monocolor, Constraints: None, Row, Col, Colour, Diagonal)
    - Pixel View (2 types - on or off

- Each agent has 3 tries for getting an answer, up to 3 environment feedback loops

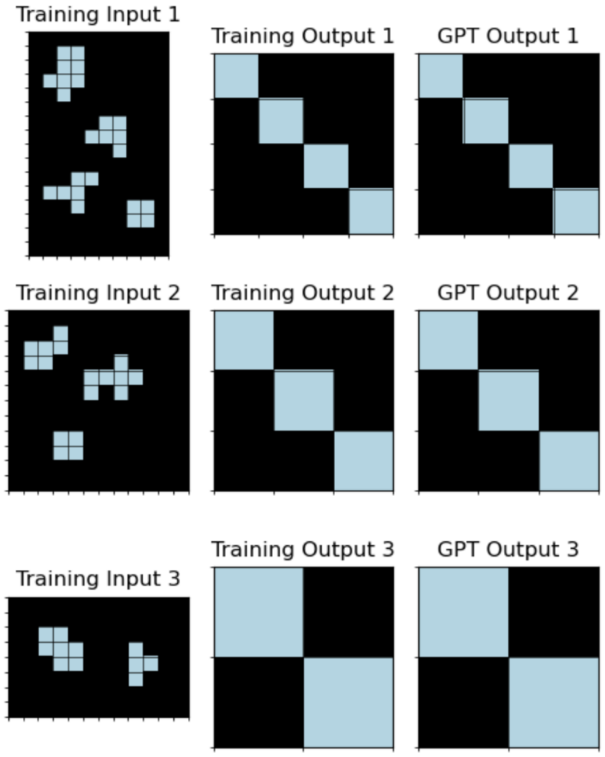
- Filter top 3 codes which passed the training set cases to try on test set

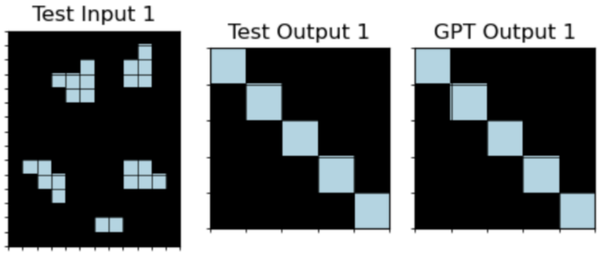
Results

- **50 out of 111 tasks solved (45%)**
  - 7 solved with iterative feedback (6 incorrect output, 1 compile error)
  - Num of Solves for Agents:
    - 7 Grid View
    - **23 Object View**
    - 19 Pixel View
    - 1 Object & Pixel View

- 61 out of 111 tasks not solved
  - 3 tasks partially solved (pass training but not test)
  - 8 tasks with correct description but wrong program output (program error)

Example: d0f5fe59, Object View





Chain of Thought (CoT) prompting via JSON

You are to output the following in json format:

{**’reflection’:** ’reflect on the answer’,

’pixel_changes’: ’describe the changes between the input and output pixels, focusing on movement or pattern changes’,

’object_changes’: ’describe the changes between the input and output objects, focusing on movement, object number, size, shape, position, value, cell count’,

’helper_functions’: ’list any relevant helper_functions for this task’,
’overall_pattern’: ’describe the simplest input-output relationship for all input-output pairs’,
’program_instructions’: ’Plan how to write the python function and what helper functions and conditions to use’,
’python_program’: "Python function named ’transform_grid’ that takes in a 2D grid and generates a 2D grid. Output as a string in a single line with \n and \t."}.

Do not use quotation marks ’ or " within the fields unless it is required for the python code

GPT Output (1/2)

- **reflection:** This task involves objects found in the grid which are simplified and moved to a certain pattern in the output grid. The input grids contain a variety of object shapes, some of which are isolated pieces while others are complex shapes. These objects, regardless of their original form, are reduced to a single cell in the output grid and relocated to the diagonal line from the top-left to the bottom-right.
- **pixel_changes:** In the transformation from input to output, the number of pixels with color ’h’ decreases. This decrease occurs because every ’h’ color object is condensed to a single pixel in the output. Also, the pixels are relocated, following a diagonal line from top left to bottom right, where the first object is placed at (0,0), the second at (1,1), and so on, regardless of their initial positions.
- **object_changes:** The number of objects remains the same from input to output. However, their size, shape, and positions change significantly. Each object, regardless of its initial shape and size, is simplified to a single pixel in the output grid. The position of the objects follows a specific pattern in the output: they are relocated to the diagonal line from the top left to the bottom right. The sequence of placement on the diagonal line is according to the order of the objects in the input grid.
- **helper_functions:** The relevant helper functions for this task are get_objects() for identifying objects in the input grid, fill_value() for placing single pixels at specific locations in the output grid, and empty_grid() for initializing the output grid.

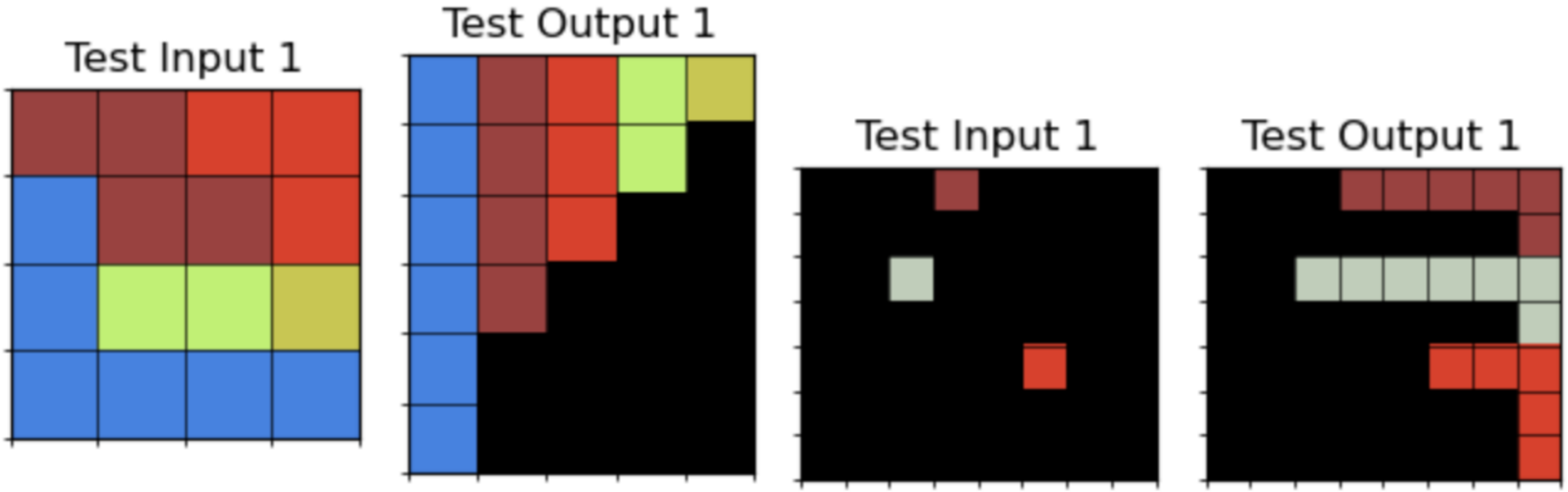
GPT Output (2/2)

- **overall_pattern:** The task takes an input grid containing a number of objects, simplifies each object into a single cell, and repositions these cells along a diagonal line from the top left to the bottom right. The order of placement on the diagonal is according to the order of the objects in the input grid. The size of the output grid is determined by the number of objects in the input grid.
- **program_instructions:**
  1. Use get_objects() function to retrieve all objects from the input grid.
  2. Create an empty grid using empty_grid(), where the number of rows and columns equals the number of objects retrieved.
  3. Loop through each object obtained from get_objects(). For each object, fill a cell in the new grid with the object color at a diagonal position corresponding to the index of the object in the objects list. This is done by calling fill_value() with the index as both the row and column position.
- **python_program:**

```
def transform_grid(grid):
    objects = get_objects(grid)
    new_grid = empty_grid(len(objects), len(objects))
    for i, obj in enumerate(objects):
        new_grid = fill_value(new_grid, (i, i), get_object_color(obj))
    return new_grid
```

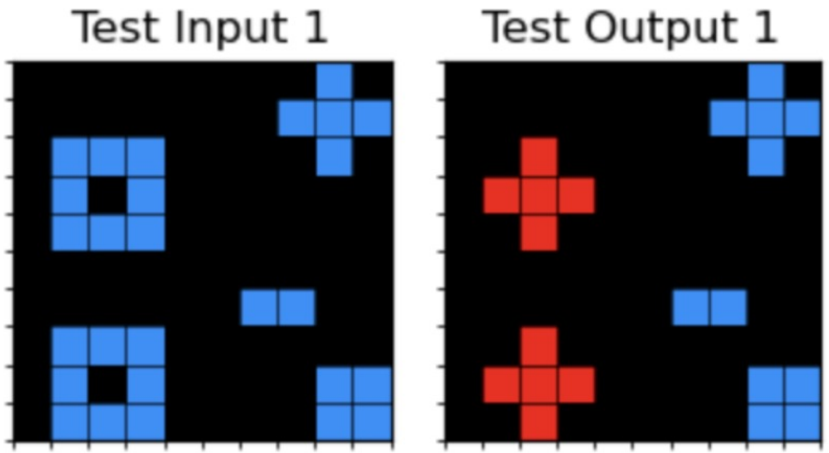
Pixel View Solves

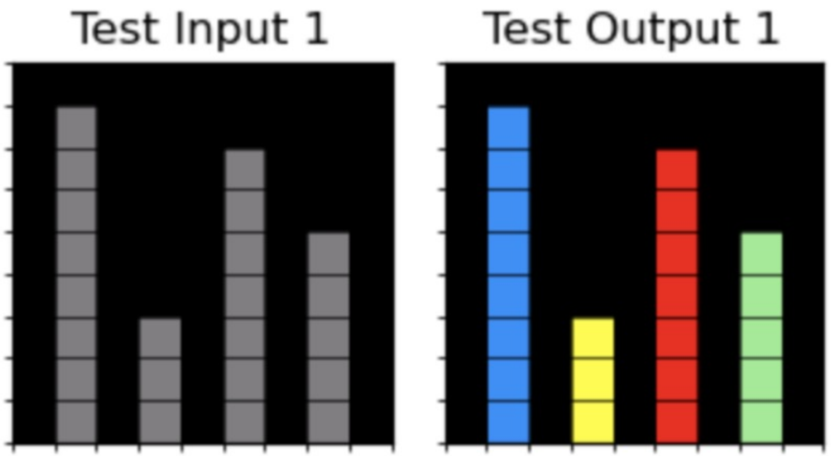
- Best for problems needing pixel coordinates
- Good for irregular shapes that have same the colour



Object View Solves

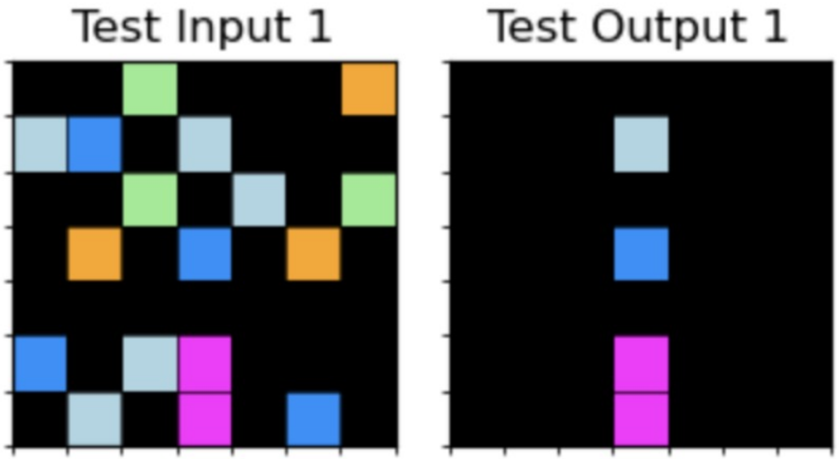
- Best for operations involving object attributes (shape, size, cell count etc.)





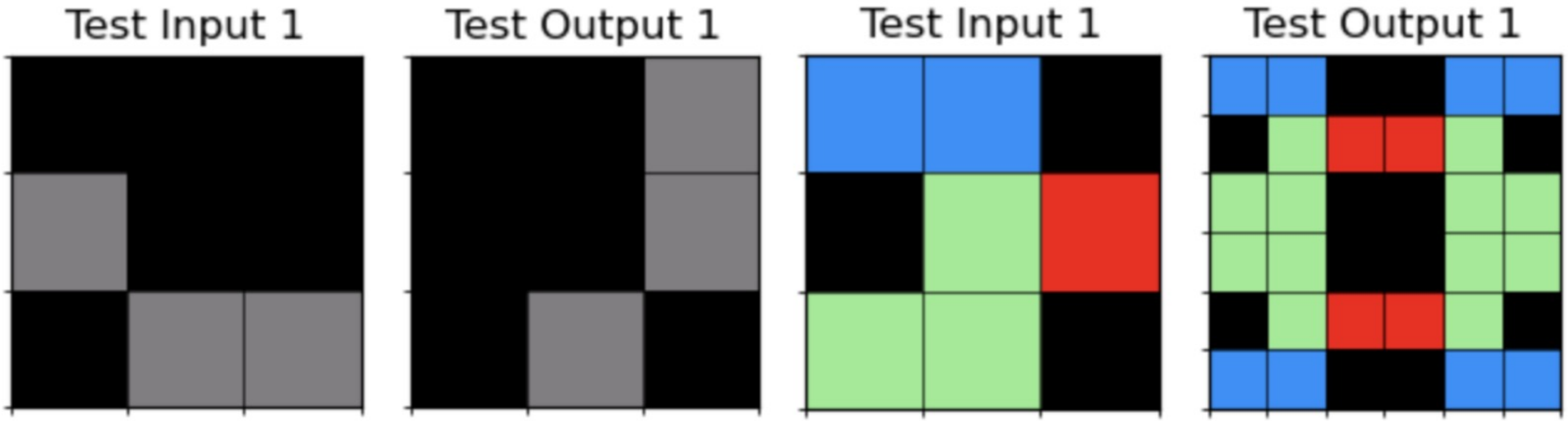
Object and Pixel View Solves

- Best for operations involving row/col and needing pixel coordinates



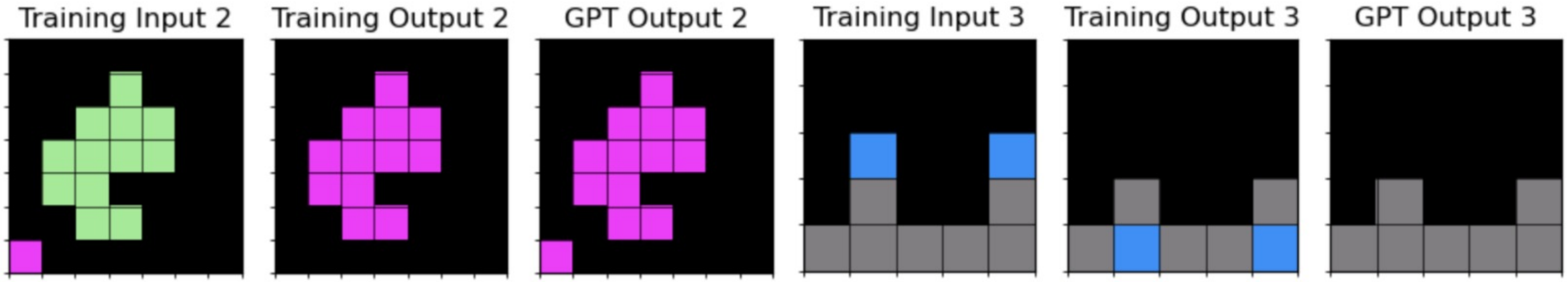
Grid View Solves

- Best for operations involving rotation / symmetry / reflection



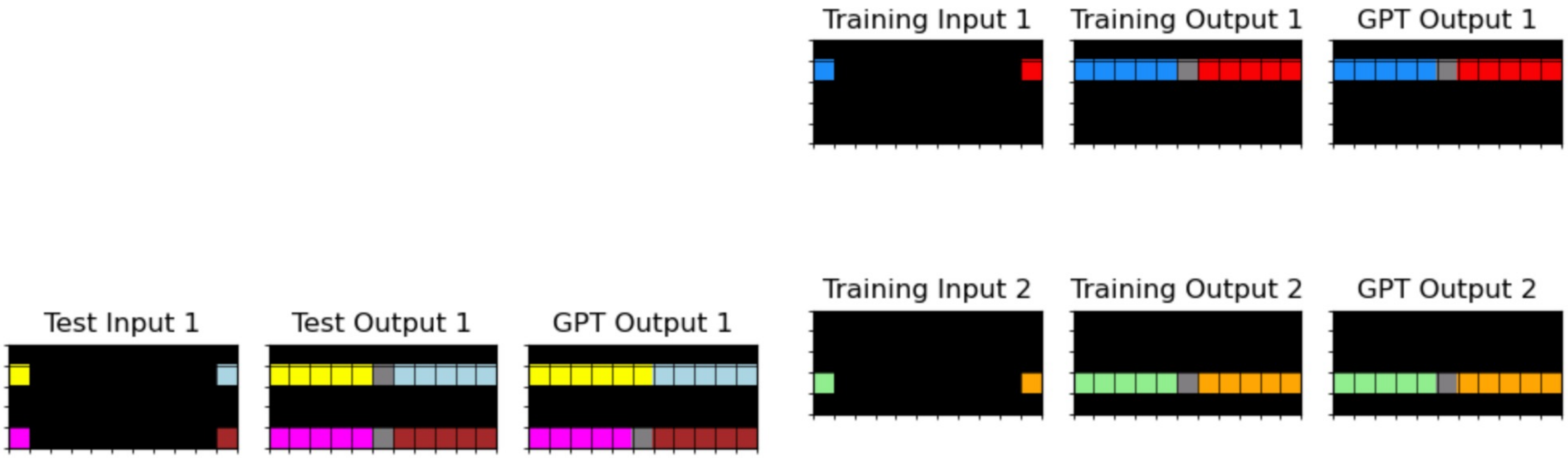
Iterative Feedback Solves

- Best for multiple operations chained together, and problem can be simplified after performing one operation

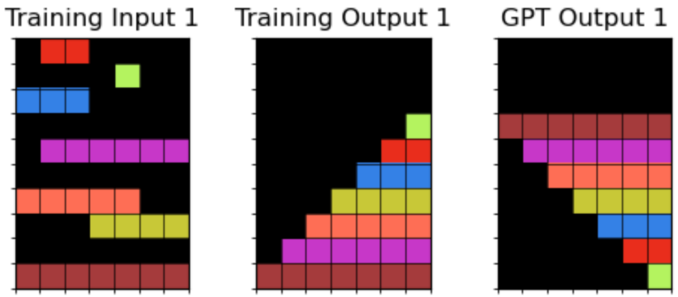


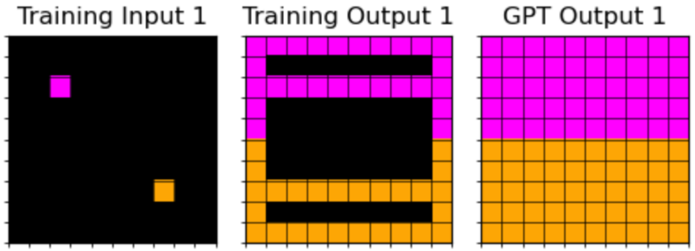
Train Pass but Test Failed

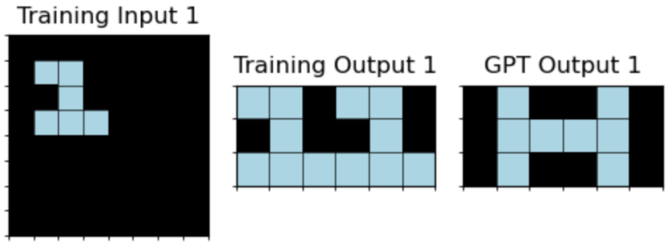
- Works well on all training set tasks, but test set includes another dimension

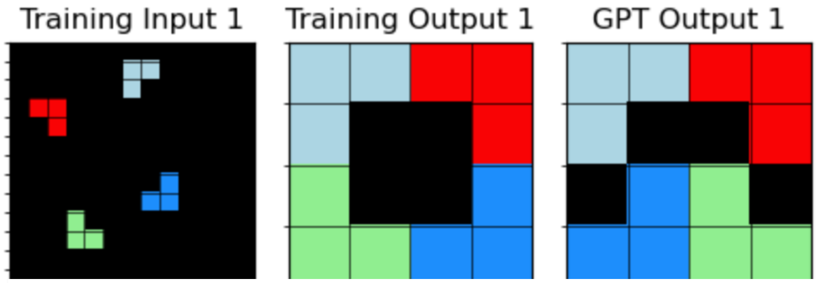


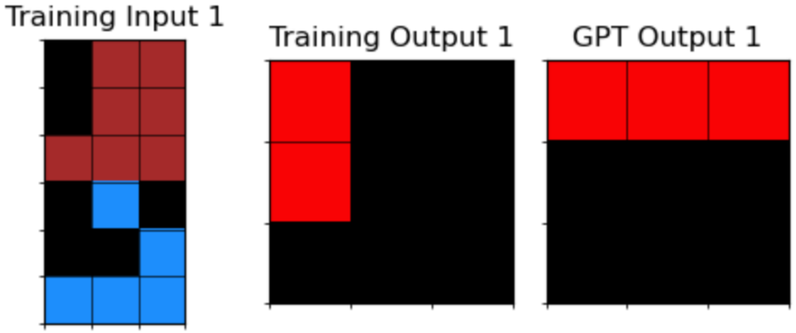
Non-Solves

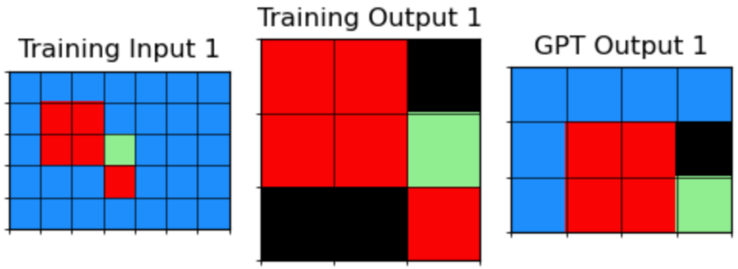

Training Input 1
Training Output 1
GPT Output 1


Training Input 1
Training Output 1
GPT Output 1


Training Input 1
Training Output 1
GPT Output 1


Training Input 1
Training Output 1
GPT Output 1


Training Input 1
Training Output 1
GPT Output 1


Training Input 1
Training Output 1
GPT Output 1

Further Improvement Ideas

Improvements

- Constrain Functions based on Agent Type

- Make Output based on Agent Type – Grid, Object, or Pixel Based Output

- Infuse more views
  - **Better Object View** – More details in Object View like bottom right coordinate
  - **Edge detection View** – Detect object features (maybe GPT-4V can help)
  - **Rotation View** – Helps rotate grid to make problem easier (row-based performs better than column-based due to next-token prediction)
  - **Diagonal View** – Rotates grid by 45 degrees to make problems become row-based
  - **Difference View** – Incorporates differences between inputs and outputs to identify where to focus on

Questions to Ponder

- Is Python code the best way to encode a program? Can there be an abstracted view like a sequence of conditional functions?
- How can we reduce the context length of the prompt?
- How can we learn useful functions via memory?
- What other views or functions are helpful? Can GPT-4V help with the ARC Challenge input visualization?

Extra Information

Prompt Details

For more details, refer to https://github.com/tanchongmin/ARC-Challenge

System Prompt

You are given a series of inputs and output pairs.

The values from ’a’ to ’j’ represent different colors. ’.’ is a blank cell.

For example, [[’.’,’a’,’.’],[’.’,’.’,’b’]] represents a 2 row x 3 col grid with color a at position (1,0) and color b at position (2,1).

Coordinates are 2D positions (row, col), row representing row number, col representing col number, with zero-indexing.

Input/output pairs may not reflect all possibilities, you are to infer the simplest possible relation.

[Helper/Primitive Functions Description + Example]

[Conditional Functions Description + Example]

User Prompt

All coordinates are given as (row,col).

Use get_size(grid) to return (len(grid),len(grid[0])).

To get objects, use get_objects(diag=False,by_row=False,by_col=False,by_color=False, multi- color=False,more_info=True) # replace this with whatever object view was used

[JSON with various abstraction views of input/output, and input and output grid size]

[Environmental Feedback – Empty, Compile Error, or Output Error Message]

Primitive/Helper Function Prompt

Each of the input-output relation can be done with one or more helper functions chained together. Some relations require other functions, which you will need to come up with yourself.

Objects are tight-fitted grids (no empty row or column) with a top left coordinate, which can be used for easy manipulation of multiple coordinates.

You can create your own objects by just creating a dictionary with ’tl’ and ’grid’

You can change an object’s position by using ’tl’ and its composition using ’grid’.

You should start each program by copying input grid or empty_grid or crop_grid of desired output size.
Then, fill the grid by using the fill helper functions.

If you use the fill functions with a ’.’ value, it is equivalent to removing parts of the grid.

[Zero-shot Description]

[One-shot Example]

Helper Function List (1/2)

- **get_objects(grid,diag=False,by_row=False,by_col=False,by_color=False,multicolor=False,more_info = True)**: Takes in grid, returns list of object dictionary: top-left coordinate of object (’tl’), 2D grid (’grid’) by_row views splits objects by grid rows, by_col splits objects by grid columns, by_color groups each color as one object, multicolor means object can be more than one color. Empty cells within objects are represented as ’$’. If more_info is True, also returns size of grid (’size’), cells in object (’cell_count’), shape of object (’shape’)
- **get_pixel_coords(grid)**: Returns a dictionary, with the keys the pixel values, values the list of coords, in sorted order from most number of pixels to least
- **empty_grid(row, col)**: returns an empty grid of height row and width col
- **crop_grid(grid, tl, br)**: returns cropped section from top left to bottom right of the grid
- **tight_fit(grid)**: returns grid with all blank rows and columns removed
- **combine_object(obj_1, obj_2)**: returns combined object from obj_1 and obj_2. if overlap, obj_2 overwrites obj_1
- **rotate_clockwise(grid, degree=90)**: returns rotated grid clockwise by a degree of 90, 180, 270 degrees
- **horizontal_flip(grid)**: returns a horizontal flip of the grid
- **vertical_flip(grid)**: returns a vertical flip of the grid
- **replace(grid, grid_1, grid_2)**: replaces all occurences of grid_1 with grid_2 in grid

Helper Function List

- **get_object_color(obj)**: returns color of object. if multicolor, returns first color only
- **change_object_color(obj, value)**: changes the object color to value
- **fill_object(grid, obj, align=False)**: fills grid with object. If align is True, makes grid same size as object
- **fill_row(grid, row_num, value, start_col=0, end_col=30)**: fills output grid with a row of value at row_num from start_col to end_col (inclusive)
- **fill_col(grid, col_num, value, start_row=0, end_row=30)**: fills output grid with a column of value at col_num from start_row to end_row (inclusive)
- **fill_between_coords(grid, coord_1, coord_2, value)**: fills line between coord_1 and coord_2 with value
- **fill_rect(grid,tl,br,value)**: fills grid from tl to br with value. useful to create rows, columns, rectangles
- **fill_value(grid, pos, value)**: fills grid at position with value

Conditional Function Prompt

Each helper function can be conditional. The conditions can be:

- by attribute, such as shape, color, position, size, cell number of object
- the condition can be an attribute on all objects, for instance, objects with the most common or least common value, or objects with the most or least common shape
- by position of pixels, such as row or column
- by neighbouring cell types or values

[Zero-shot Description]

[One-shot Example]

Conditional Function List

- **object_contains_color(obj, value)**: Returns True/False if object contains a certain value

- **on_same_line(coord_1, coord_2)**: Returns True/False if coord_1 is on the same line as coord_2. line_type can be one of [’row’, ’col’, ’diag’]