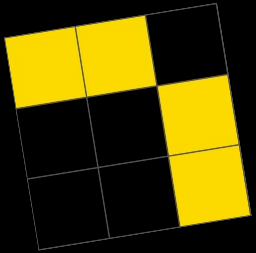
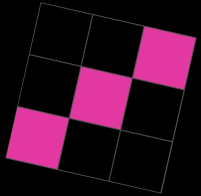



intro2ai

Big Model Cowboycraft:
Visual Reasoning & Wranglin’ GPTs

Jake Peterson | Emmit Renick | Lucas Whitbey

Assignment Memo

Nervous? Don’t be.

No coding experience necessary, no PhD required, and absolutely no AI expertise expected.

You see, you can make magic.

And we’re going to show you how. Rodeo style.

Along the way, you’ll learn to approach problems systematically, identify patterns, and communicate your ideas effectively.

It’s all in the Cowboycraft.

You will learn how AI thinks.
You will learn its strengths & weaknesses.

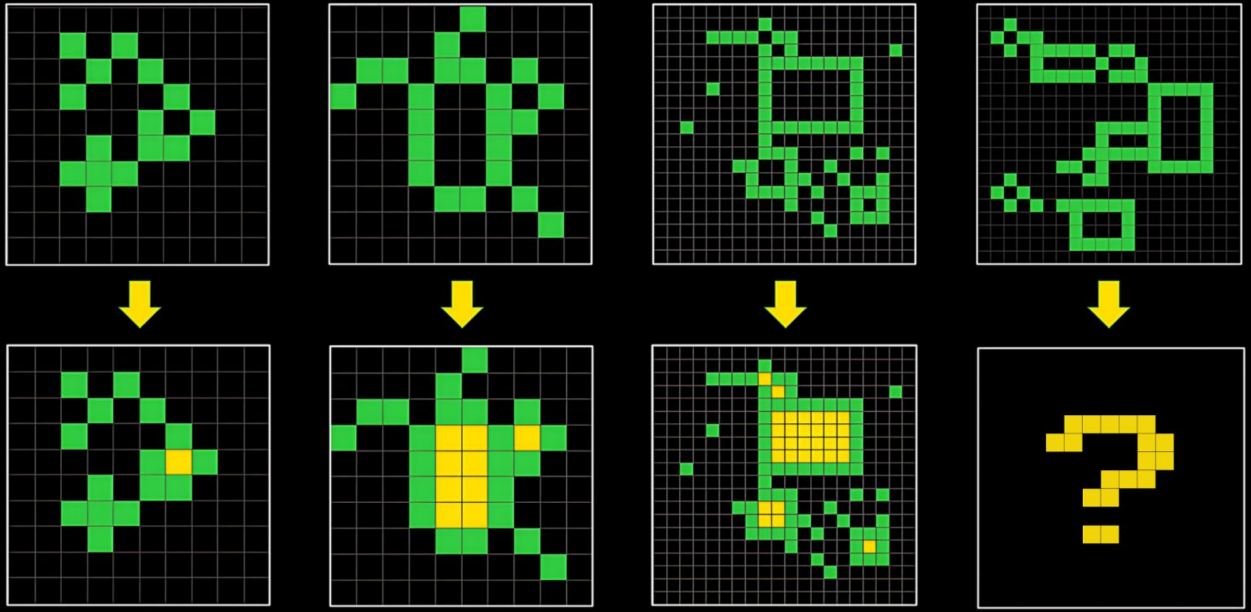
And then, you will wrangle, rope, wrestle, & finesse
ChatGPT ‘til it dances any which way you please.

ChatGPT is a Large Language Model.

Trained on a GIGANTIC corpus of text from anywhere and everywhere. It’s really good at almost everything ever publicly documented in text.

But it sucks at visual reasoning.

Even the easy stuff. Like this:



Step 1: Get Your Hands Dirty

Visit https://arcprize.org/play and try out the daily puzzle. Don’t worry about being perfect—just explore and see what patterns you can find.

1. What’s happening in the puzzle?
2. Can you spot any patterns? Shapes or colors?
3. What changes from the input to the output?
4. Try a few more by selecting from the dropdown menu.

Step 2: Reflect on Your Thinking

Take a moment to think about
how you naturally notice patterns.

1. Do you see objects in the grid? (Shapes or groups that feel connected.)
2. Are you focusing on how colors or lines stand out?
3. Are there clear relationships? Groups? Stacks? Connections? Objects?
4. What helps you spot patterns quickly?
5. How do you decide what’s important to notice?

Step 3: Articulate your thinking

The skills you use without even thinking are powerful tools!

These basic ideas—shapes, colors, groups, positions—are your human “toolkit” for solving problems. However, we often poorly describe them. As humans, we all, “know what you mean.”

1. Explain your thought process to a team member.
2. Try to describe what the core skills are, and how they relate.
3. Can you explain what an “object” is?
4. Or why one square feels more important than another?

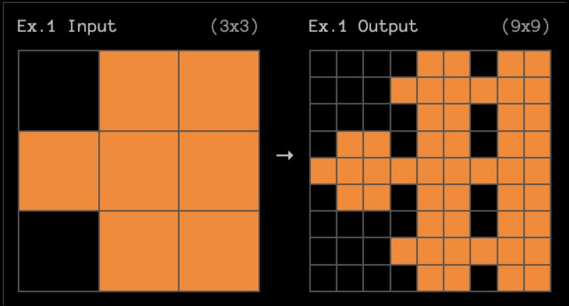
Step 4: Study Your Task

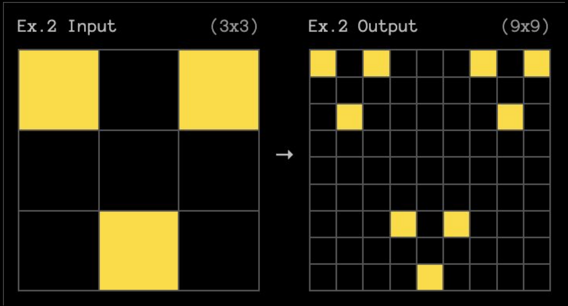
Get your task from the problem generator
and study the "before" and "after" grids.

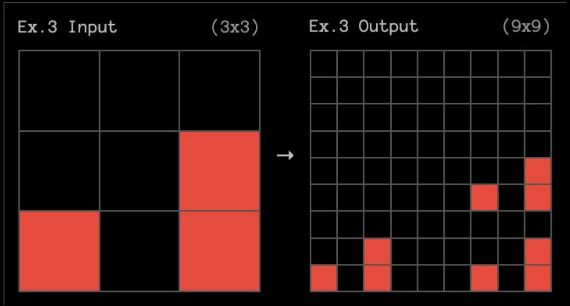
(or use the problem on the next slide)

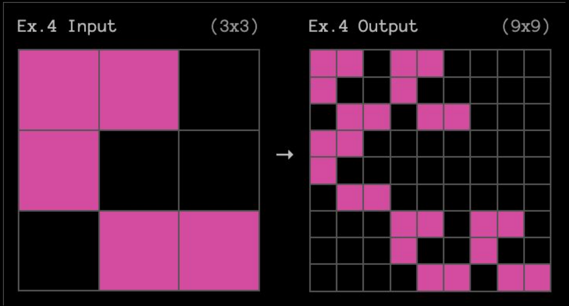
1. What’s in the grid? Are there shapes, colors, lines, or empty spaces?
2. What changes from the input to the output? Positions? Colors?
3. Can you group parts of the grid?
4. Are there clusters that seem connected?
5. How are the pieces related? Do they align, mirror, or repeat?

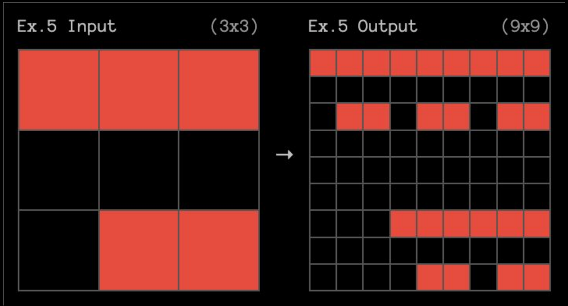
Examples:











https://arcprize.org/play?task=007bbfb7

Step 5: Break It Down

Try to describe what is happening in english. Lay out the rules that govern the transformation and the steps that change the input to the output.

1. Which adverbs and adjectives do you use to describe objects and change?
2. Can you think of a way a computer could “sense” or “detect” the same?
3. Can you describe the rules in numbers or positions, rows or columns?

Step 6: Describe the Big Steps

Look at your solution and break it into big, overarching steps. Imagine you’re explaining what to do, one major task at a time, to someone who has no context. Write these steps in clear, plain English.

For example:

1. Make a new grid the same size as the input.
2. Identify all yellow triangles.
3. Rotate each yellow triangle 90 degrees clockwise.
4. Place a blue square at the tip of each vertex.

Step 7: Describe the Substeps

Now take each of your big steps and break it into smaller, detailed substeps. This is where you explain exactly how to do each part.

For example:
Big Step: Identify all yellow triangles.
Substeps:

1. Look at each square in the grid, one by one.
2. If a square is yellow mark it.
3. For all yellow squares, check neighbor squares for yellow squares.
4. For all neighbor groups of yellow squares… etc.

Step 8: Test It Out

Now read your steps to a partner. Hand them a screenshot of your grid and have them follow your steps exactly. No guessing or adjustments.

1. Did the instructions work perfectly?
2. Were there any moments where your partner got confused?
3. Did they ever have to guess what you meant?
4. Was it too specific? (only working for one example)
5. Or too vague? (missing key details)

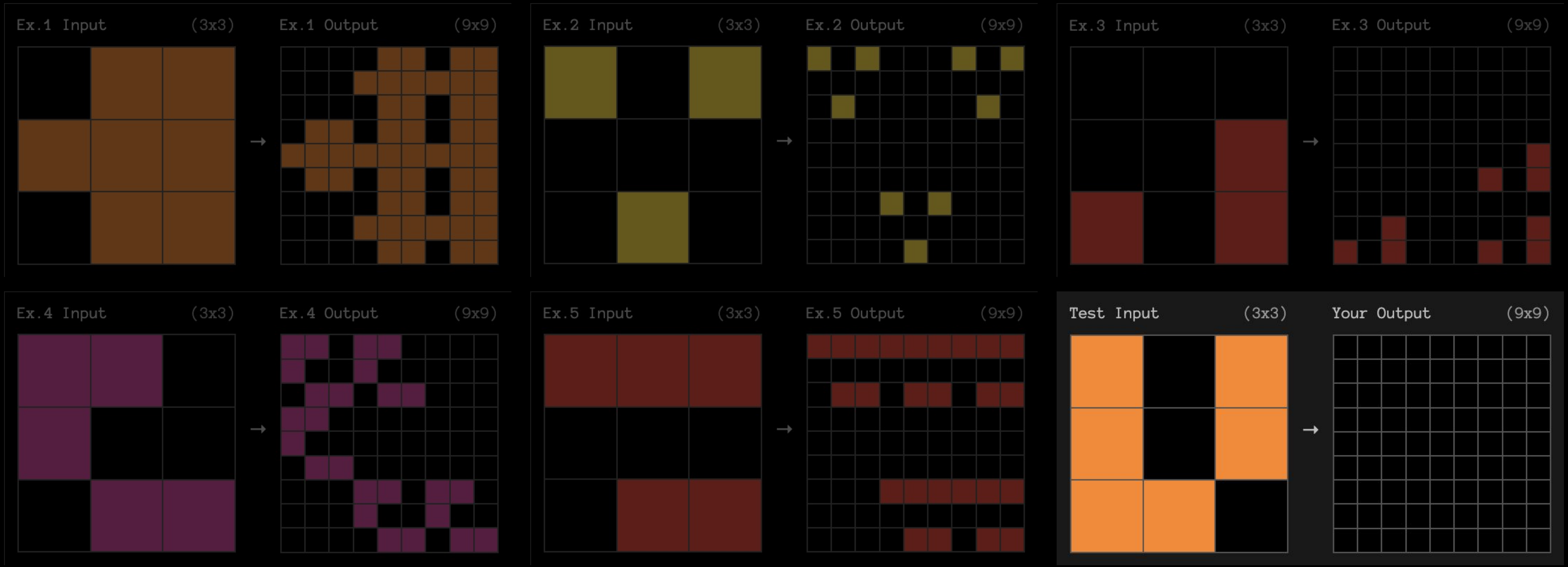
Step 8B: Test It Out

After refining your instructions, check out the hidden test grid. Does your solution still work?

(Check out the next slide for the hidden test)

1. Why did your solution work/fail on the new test input?
2. How can you refine the steps to avoid confusion?
3. How could you adjust the rules to handle every case?
4. What are the strongest/weakest aspects of your solution?

The Test:



https://arcprize.org/play?task=007bbfb7

Step 9: Take It to the Rodeo

Take your step-by-step instructions and ask ChatGPT (or another AI) to write Python code to execute them. Test the code with your puzzle examples.

1. Did the code match your instructions?
2. If it didn’t work, were your steps unclear?
3. Or did the AI misunderstand?

If the code doesn’t work, tweak your instructions so the AI can understand you. Machines don’t read minds—they only do exactly what they’re told.

Step 10: Dream On Cowboy

Look back at the puzzle and your process. Notice iteration and communication’s role in collaboration.

1. How did you figure out what was important?
2. Did the back-and-forth nature of refinement help?
3. How did you identify gaps in understanding?
4. How could you teach the AI to solve similar puzzles?

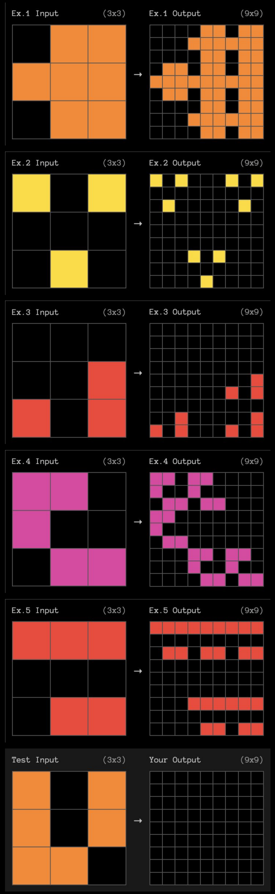
Congrats. You’re a certified Big-Model-AI-Cowboy.

What will you do next?

Annotated Example

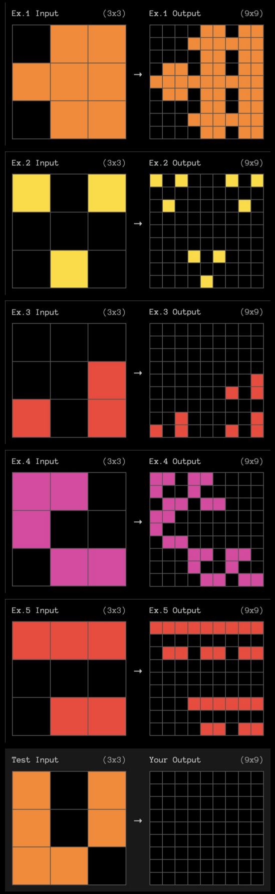
Original English Description:

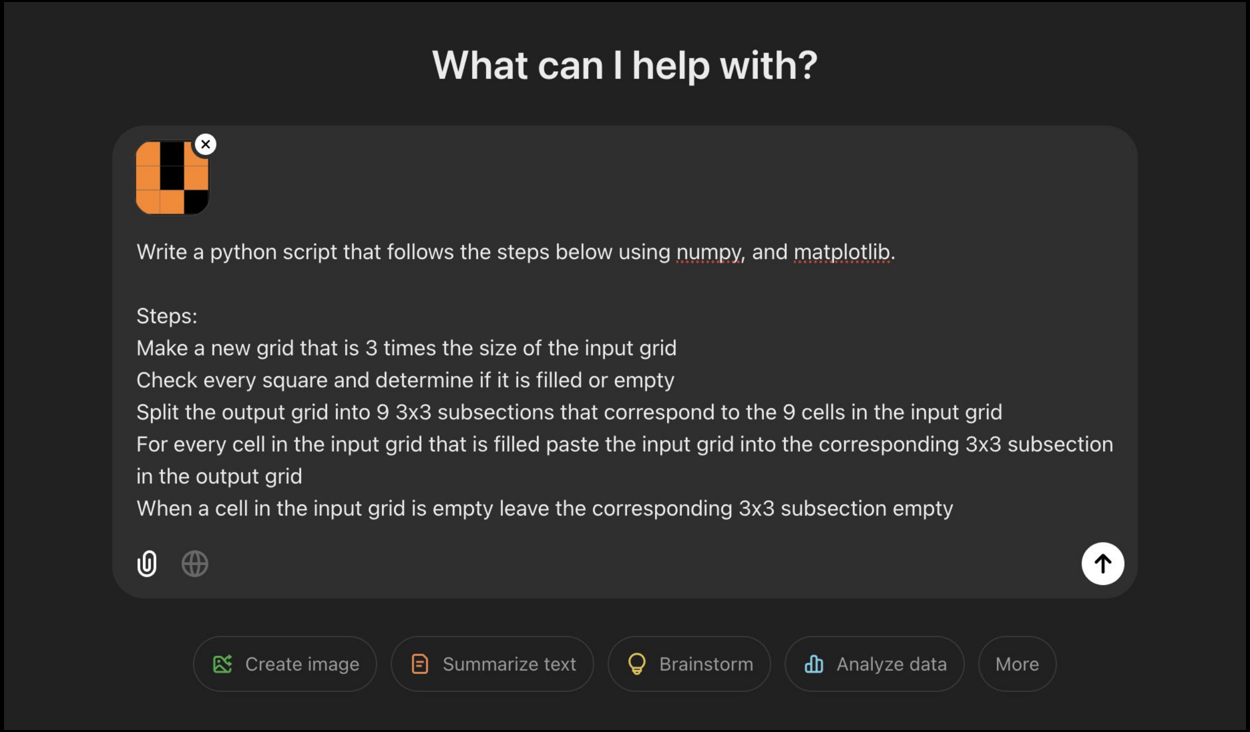
The input image is transformed from a 3 by 3 grid into a 9 by 9 grid.The 9 by 9 grid (output grid) is then split into 9, 3 by 3 subsections. For every corresponding section that has color on the input grid, the output grid simply puts the whole pattern in that subsection.



Original Steps in English:

1. Make a new grid that is 3 times the size of the input grid
2. Check every square and determine if it is filled or empty
3. Split the output grid into 9 3x3 subsections that correspond to the 9 cells in the input grid
4. For every cell in the input grid that is filled paste the input grid into the corresponding 3x3 subsection in the output grid
5. When a cell in the input grid is empty leave the corresponding 3x3 subsection empty





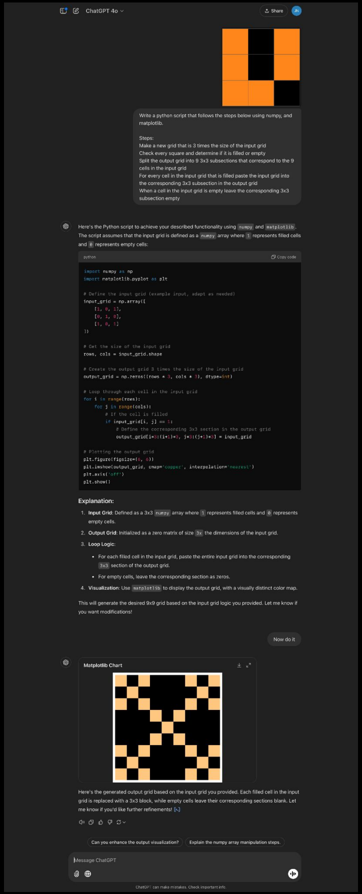
What can I help with?

Write a python script that follows the steps below using numpy, and matplotlib.

Steps:
Make a new grid that is 3 times the size of the input grid
Check every square and determine if it is filled or empty
Split the output grid into 9 3x3 subsections that correspond to the 9 cells in the input grid
For every cell in the input grid that is filled paste the input grid into the corresponding 3x3 subsection in the output grid
When a cell in the input grid is empty leave the corresponding 3x3 subsection empty

ChatGPT Generated Code 1:

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the input grid (example input, adapt as needed)
input_grid = np.array([
    [1, 0, 1],
    [0, 1, 0],
    [1, 0, 1]
])

# Get the size of the input grid
rows, cols = input_grid.shape

# Create the output grid 3 times the size of the input grid
output_grid = np.zeros((rows * 3, cols * 3), dtype=int)

# Loop through each cell in the input grid
for i in range(rows):
    for j in range(cols):
        # If the cell is filled
        if input_grid[i, j] == 1:
            # Define the corresponding 3x3 section in the output grid
            output_grid[i*3:(i+1)*3, j*3:(j+1)*3] = input_grid

# Plotting the output grid
plt.figure(figsize=(6, 6))
plt.imshow(output_grid, cmap='copper', interpolation='nearest')
plt.axis('off')
plt.show()
```



Write a python script that follows the steps below using numpy, and matplotlib.

Steps:
Make a new grid that is 3 times the size of the input grid
Check every square and determine if it is filled or empty
Split the output grid into 9 3x3 subsections that correspond to the 9 cells in the input grid
For every cell in the input grid that is filled paste the input grid into the corresponding 3x3 subsection in the output grid
When a cell in the input grid is empty leave the corresponding 3x3 subsection empty

Here's the Python script to achieve your described functionality using `numpy` and `matplotlib`. The script assumes that the input grid is defined as a `numpy` array where `1` represents filled cells and `0` represents empty cells:

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the input grid (example input, adapt as needed)
input_grid = np.array([
    [1, 0, 1],
    [0, 1, 0],
    [1, 0, 1]
])

# Get the size of the input grid
rows, cols = input_grid.shape

# Create the output grid 3 times the size of the input grid
output_grid = np.zeros((rows * 3, cols * 3), dtype=int)

# Loop through each cell in the input grid
for i in range(rows):
    for j in range(cols):
        # If the cell is filled
        if input_grid[i, j] == 1:
            # Define the corresponding 3x3 section in the output grid
            output_grid[i*3:(i+1)*3, j*3:(j+1)*3] = input_grid

# Plotting the output grid
plt.figure(figsize=(6, 6))
plt.imshow(output_grid, cmap='copper', interpolation='nearest')
plt.axis('off')
plt.show()
```

Explanation:

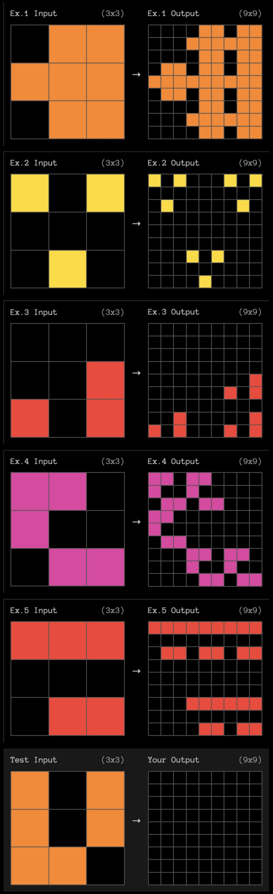
1. **Input Grid**: Defined as a 3x3 `numpy` array where `1` represents filled cells and `0` represents empty cells.
2. **Output Grid**: Initialized as a zero matrix of size `3x` the dimensions of the input grid.
3. **Loop Logic**:
   - For each filled cell in the input grid, paste the entire input grid into the corresponding `3x3` section of the output grid.
   - For empty cells, leave the corresponding section as zeros.
4. **Visualization**: Use `matplotlib` to display the output grid, with a visually distinct color map.

This will generate the desired 9x9 grid based on the input grid logic you provided. Let me know if you want modifications!

Now do it

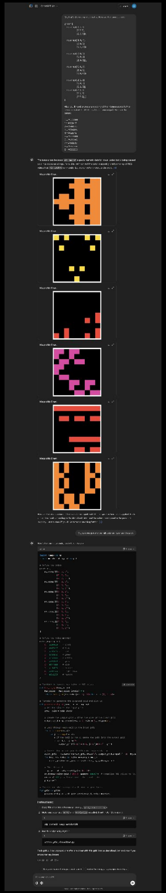
Here's the generated output grid based on the input grid you provided. Each filled cell in the input grid is replaced with a 3x3 block, while empty cells leave their corresponding sections blank. Let me know if you'd like further refinements! [>_]

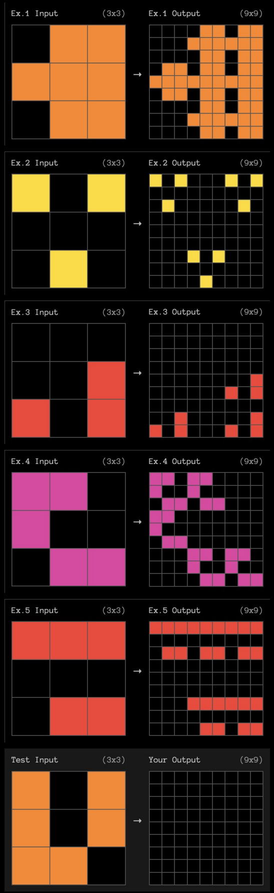
Can you enhance the output visualization?
Explain the numpy array manipulation steps.
Message ChatGPT
ChatGPT can make mistakes. Check important info.



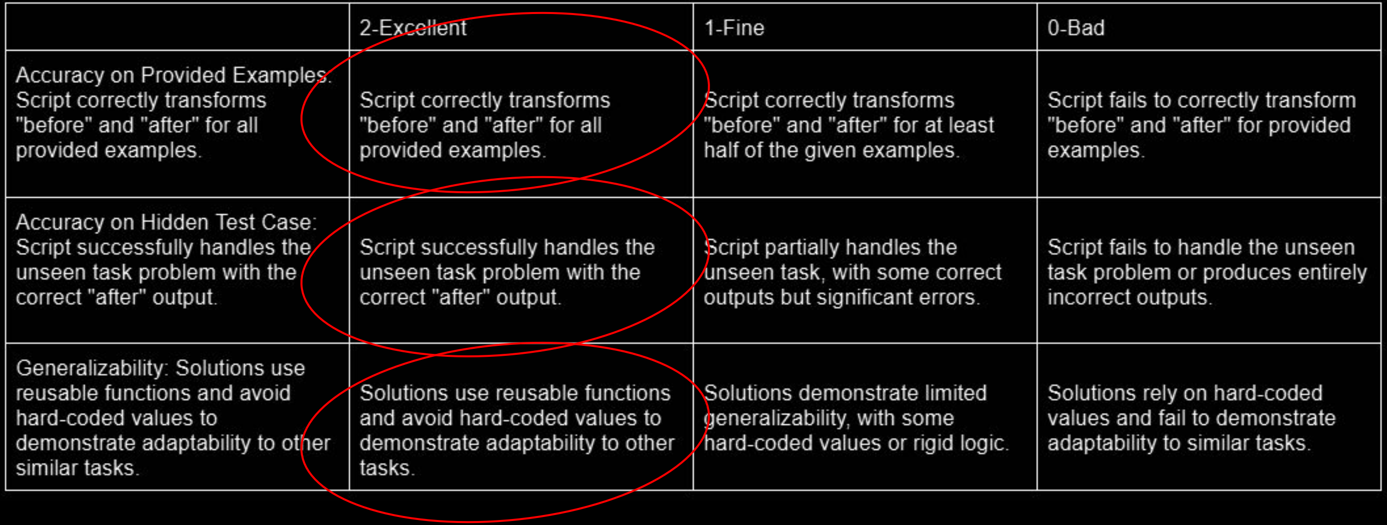
Refined ChatGPT Generated Code 2:

```
# Function to convert hex color to RGB values
def hex_to_rgb(hex_color):
    hex_color = hex_color.lstrip('#')
    return tuple(int(hex_color[i:i+2], 16) for i in (0, 2, 4))

# Function to generate the expanded grid and plot it
def process_and_plot(grid, color_mapping):
    # Get the size of the input grid
    rows, cols = grid.shape

    # Create the output grid 3 times the size of the input grid
    output_grid = np.zeros((rows * 3, cols * 3), dtype=int)

    # Loop through each cell in the input grid
    for i in range(rows):
        for j in range(cols):
            # If the cell is not 0, paste the grid into the output grid
            if grid[i, j] != 0:
                output_grid[i*3:(i+1)*3, j*3:(j+1)*3] = grid

    # Convert the output grid to RGB color representation
    color_grid = np.zeros((output_grid.shape[0], output_grid.shape[1], 3), dtype=int)
    for key, hex_color in color_mapping.items():
        color_grid[output_grid == key] = hex_to_rgb(hex_color)

    # Plot the grid
    fig, ax = plt.subplots(figsize=(6, 6))
    ax.imshow(color_grid / 255.0, aspect='equal')  # Normalize RGB values to [0, 1]
    ax.axis('off')  # Remove axis labels and ticks
    plt.show()

# Process and plot each grid
for grid in grids:
    process_and_plot(grid, color_mapping)
```





Grading



	2-Excellent	1-Fine	0-Bad
Accuracy on Provided Examples: Script correctly transforms "before" and "after" for all provided examples.	Script correctly transforms "before" and "after" for all provided examples.	Script correctly transforms "before" and "after" for at least half of the given examples.	Script fails to correctly transform "before" and "after" for provided examples.
Accuracy on Hidden Test Case: Script successfully handles the unseen task problem with the correct "after" output.	Script successfully handles the unseen task problem with the correct "after" output.	Script partially handles the unseen task, with some correct outputs but significant errors.	Script fails to handle the unseen task problem or produces entirely incorrect outputs.
Generalizability: Solutions use reusable functions and avoid hard-coded values to demonstrate adaptability to other similar tasks.	Solutions use reusable functions and avoid hard-coded values to demonstrate adaptability to other tasks.	Solutions demonstrate limited generalizability, with some hard-coded values or rigid logic.	Solutions rely on hard-coded values and fail to demonstrate adaptability to similar tasks.

Let’s Brag:

“I think the topic we picked was very interesting and we got a lot of work done on the project.”

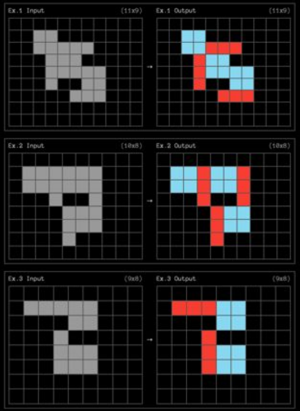
Let’s Be Critical:

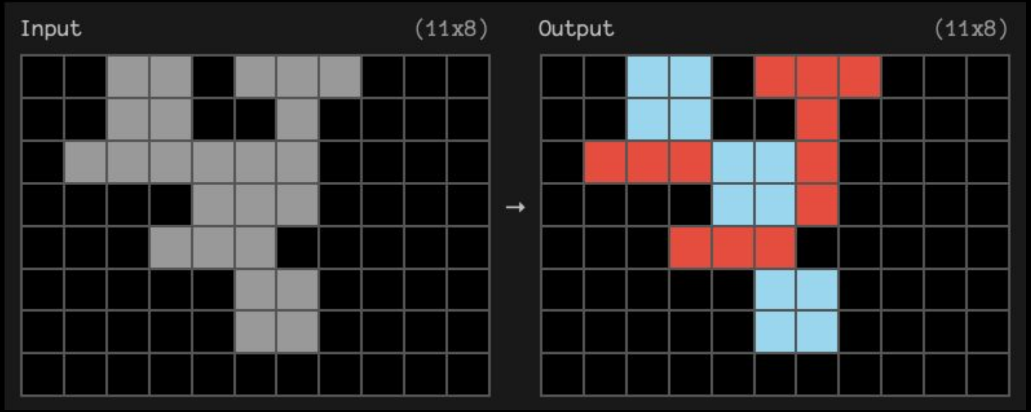
“We had trouble getting the code to be a perfect example because it was hard to create a generalized solution.”

Additional Examples

Test Example 1

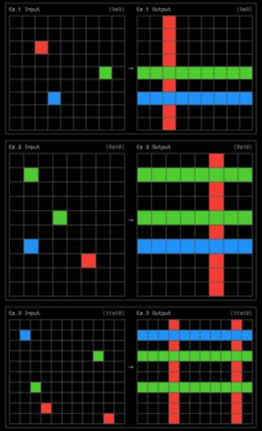
1. The input grid has dimensions 11x8.
2. The grid has connected 1x1 grey squares, which form shapes that we need to analyze.
3. Copy the input shape into the output side.
4. Identify as many 2x2 squares as possible and replace those cells with blue. Next, cover the remaining grey cells with 3x1 red rectangles.
5. The result should have an 11x8 grid with blue 2x2 squares and 3x1 rectangles that replicates the input shape. There should be no leftover grey cells.

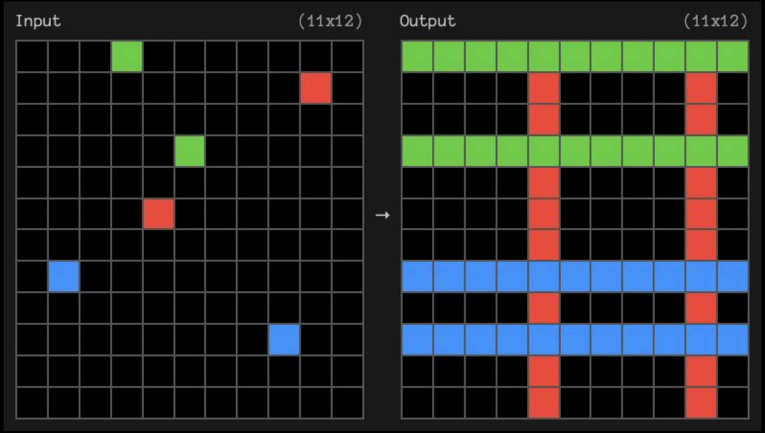




Test Example 2

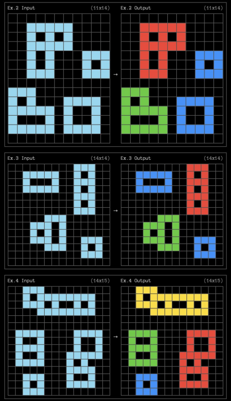
1. The input grid has dimensions 11x12.
2. There are 1x1 squares that are either red, blue, or green on the input side.
3. If there's a red square on the input side, that entire column is colored red.
4. If there's a blue/green square on the input side, that entire row is colored in blue/green and covers red squares in their path.
5. Follow the pattern on the output side to solve the puzzle.

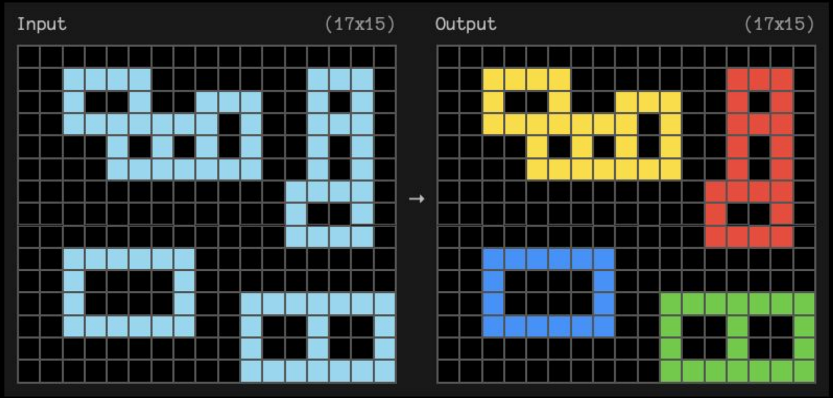




Test Example 3

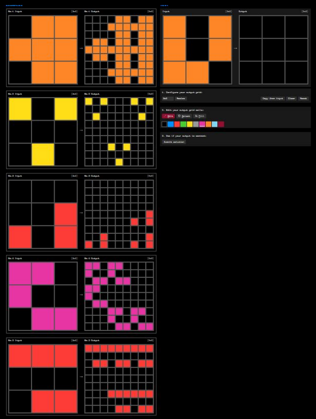
1. The input grid has dimensions 17x15.
2. There are randomly generated shapes with empty spaces.
3. The number of holes in each shape affects the color of the shape, but the size of the holes does not.
4. Shapes with one hole are colored blue, shapes with two holes are colored green, shapes with three holes are colored red, shapes with four holes are colored yellow.
5. Copy input shapes into the output side and color them according to the number of holes.

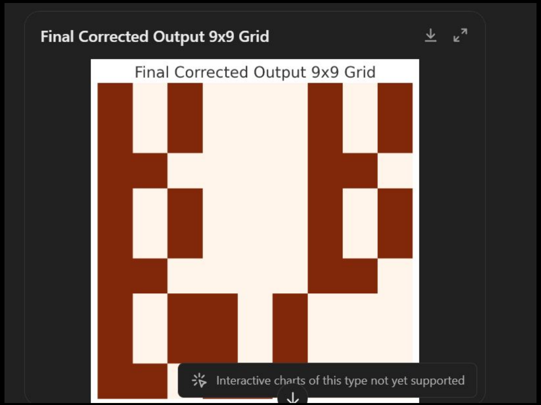


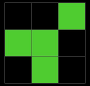


Test Example 4

1. Make a new grid that is 3 times the size of the input grid
2. Check every square and determine if it is filled or empty
3. Split the output grid into 9 3x3 subsections that correspond to the 9 cells in the input grid
4. For every cell in the input grid that is filled paste the input grid into the corresponding 3x3 subsection in the output grid
5. When a cell in the input grid is empty leave the corresponding 3x3 subsection empty




Final Corrected Output 9x9 Grid
Final Corrected Output 9x9 Grid
Interactive charts of this type not yet supported



The End!