CodePercept: Code-Grounded Visual STEM Perception for MLLMs

Supplementary Material

A. Unified Quality Control

To ensure the reliability and consistency of our data generation pipeline, we implement a unified quality control framework with explicit prompts for each stage. Our quality control operates at three critical checkpoints: (1) verifying the correctness and quality of generated code, (2) verifying the correctness and quality of rendered image, and (3) verifying the consistency of image-code pairs. Below, we present the detailed prompts used at each stage to maintain high standards of both technical accuracy and instructional effectiveness.

Code Quality Prompt $\mathcal{Q}_\mathrm{C}$

You are a leading AI scientist tasked with curating a training dataset for an Image-to-Code model designed to understand and generate expert-level human code. Your role combines philosophical insight with engineering rigor as a quality assessor.

1. ***Core Philosophy: Maximize Semantic Value***

   ***Objective**: Cultivate the model's ability to map visual patterns to high-level programming concepts by distinguishing meaningful abstractions from arbitrary procedures.*

   1.1 Conceptual Framework

   1. ***High-Value Abstraction** (samples we pursue):*
      - *Code implements mathematical concepts or algorithms with clear, visually recognizable structural features*
      - *Procedural steps are necessary components for realizing the abstraction*
      - *Examples: $y = x^2$ (parabola), Fibonacci spiral algorithms, `noise.pnoise2` (procedural noise)*
   2. ***Low-Value Procedure** (samples we reject):*
      - *Code contains arbitrary, context-free computational steps not inferable from the final image*
      - *Contains "magic numbers" and "black-box operations" toxic to model learning*
      - *Examples: `data += 100`, `data = data * 1.1`, `if unseen_variable: ...`*

2. ***Evaluation Rules***

   2.1 Pattern Recognition Rule

   1. ***[Qualified]** Image exhibits strong, non-random structural patterns:*
      - *Symmetry, spirals, fractals, waveforms, regular grids*
      - *Code clearly implements mathematical formulas or algorithms producing the pattern*
      - *This represents the highest-value sample type*
   2. ***[Qualified]** Image appears random but uses domain-standard tools:*
      - *Examples: noise maps generated via `noise.pnoise2`*
      - *Teaches proper use of professional libraries*
   3. ***[Qualified]** Simple graphics with hardcoded constants:*
      - *Bar charts, scatter plots with direct data declaration (`data = [1,2,3]`)*
      - *Basic but acceptable "declarative" samples*
   4. ***[Disqualified]** True random generation without structure:*
      - *Generated by `np.random.rand()` with no recognizable macroscopic patterns*

   2.2 Computational Intent Rule

   1. ***[Qualified]** Computational steps are intrinsic to high-value abstraction:*
      - *Example: Z-value-based color/linewidth computation in torus knot code implementing "pseudo-3D depth perception" (a universal rendering technique)*
   2. ***[Disqualified]** Arbitrary, isolated numerical operations:*
      - *Logic or context completely lost in the image*
      - *Example: `aqi_data[0, :] += 100`*

   2.3 Dependency & Source Rule

   1. ***[Qualified]** Dependencies limited to:*
      - *Python standard library*
      - *Well-known pip-installable libraries directly related to visual generation (`numpy`, `scipy`, `noise`, etc.)*

Code Quality Prompt $\mathcal{Q}_\text{C}$ (continued)

2. ***[Disqualified]*** *Reliance on inaccessible external resources:*
   - *External files (.csv, .json), databases, or network APIs*
   - *Model cannot access these resources during inference*

3. ***Required Output Format***

- ***[Verdict]***: *(Qualified / Disqualified)*
- ***[Suitability Score]***: *(1-5, where 1=harmful, 3=baseline qualified, 5=exceptionally educational)*
- ***[Rationale]***: *Briefly explain the verdict. Explicitly identify whether the code represents* ***[High-Value Abstraction]*** *or* ***[Low-Value Procedure]****. If qualified, articulate the "core concept" being taught (e.g., "recognizing Fibonacci spiral algorithms" or "learning pseudo-3D rendering techniques"). If disqualified, specify which computational step is "arbitrary and non-invertible."*

Evaluate the provided Python visualization code according to these principles.

Image Quality Prompt $\mathcal{Q}_\text{I}$

You are a professional image quality analyst. Your mission is to conduct a quality review of a given code-generated image according to the strict standards defined below. Your report must not only include the final verdict but, more importantly, ***must demonstrate a clear, systematic analysis process*** *to prove that your conclusion is well-considered.*

1. ***Stage 1: Technical Rendering Audit***

Task: *Assess whether the image has fatal technical rendering defects.*

Error Classification Definitions

A. (Class) ***Fatal Rendering Errors***: *Technical rendering failures caused by algorithmic or data interpretation failures that* ***severely compromise*** *the geometry, paths, or annotations of the graphic.*

- ***A-1. Vector Path Catastrophe***:
  - *Path Distortion: Paths exhibit* ***severe*** *unexpected spikes, kinks, or non-smoothness.*
  - *Projection Artifacts: A portion of a path or fill is* ***severely stretched*** *to or beyond the canvas boundary.*
  - *Unexpected Unclosed Path: A shape that should be closed has a* ***significant gap****, causing the fill color to "leak".*
- ***A-2. Geometric Annotation Error***:
  - *Severe Anchor Misplacement: The anchor point of an annotation element (e.g., angle arc, dimension arrow) is* ***completely detached*** *from the object it is supposed to annotate.*
- ***A-3. Empty Chart***:
  - *The coordinate system and labels are drawn correctly, but the* ***data series*** *(e.g., lines, bars) is* ***entirely missing****.*
- ***A-4. Indiscernible Density***:
  - *Contains a large number of elements that are indescribable, chaotic, indistinguishable, densely overlapping, or blurry (e.g., a "point cloud soup" or a "sea of lines").*
- ***A-5. Layout Anomaly***:
  - *Abnormal layout resulting in large areas of whitespace.*
  - *Crowded layout making key elements difficult to distinguish.*

B. (Class) ***Non-Fatal Design Flaws***: *The technical rendering is successful, but there are flaws in aesthetics or content logic.*

- *Examples: Poor color choices, normal occlusion between elements, unintuitive logical layer order, questionable content that is drawn correctly.*

2. ***Core Principles & Output Format***

- ***Principle 1: Identify Class A Errors (Critical)****. If* ***at least one significant, unintentional, technical Class A rendering error*** *is present, the verdict is* ***Fail****.*
- ***Principle 2: Exempt All Minor Issues****.* ***Extremely minor technical glitches*** *(e.g., single-pixel offsets) and* ***all Class B design flaws*** *(e.g., aesthetic issues), regardless of their number,* ***must never*** *be the basis for a "Fail" verdict.*

Output Format: *Strictly follow the process above, analyzing step-by-step and providing reasons. Then, generate the complete analysis report using the specified Markdown format below.*

```
## Rendering Quality Audit
* Final Verdict:  [Enter ``Pass" or ``Fail"]
```

Image Quality Prompt $\mathcal{Q}_{\mathrm{I}}$ (continued)

```
### Error Analysis
[Note: This section should only be generated if the final verdict is ``Fail". If
multiple errors exist, repeat the list item.]
* Error Type: [Enter A-1, A-2, etc.]
* Description: [Describe the specific error phenomenon and its location in the
image.]
```

Image-Code Consistency Prompt $\mathcal{Q}_{\mathrm{IC}}$

You are a professional image quality analyst. Your mission is to conduct a quality review of the given code-image pair according to the strict standards defined below.

Task: Determine if the **core structure and scene** presented in the image are fundamentally consistent with the **design intent** of the code.

1. ***Workflow***
   1. ***Visual Evidence Inventory***: ***Ignore the code*** *and visually inventory the elements in the image (quantity, color, layout, relationships).*
   2. ***Code Logic Deduction (Design Blueprint)***: ***Mentally compile the code****, analyzing variables (e.g.,* `n=8`*), loops (e.g.,* `for i in range(8)`*), and control flow to precisely deduce the image's* ***quantitative specifications*** *(e.g., total number of elements, exact coordinates, connection relationships, etc.).*
   3. ***Critical Comparison***: *Strictly compare the "Visual Evidence" with the "Design Blueprint".*
2. ***Core Verdict Standard: "Fundamental Mismatch"***
   - ***"Sufficient Match"***: *The* ***core elements, deduced quantities, attributes, and spatial/structural relationships*** *described by the code are fundamentally consistent with the image. Minor rendering differences (e.g., color shades, anti-aliasing effects) are permissible.*
   - ***"Fundamental Mismatch"***: *There is a* ***structural or categorical*** *essential difference between the code's intent and the image's reality. Any of the following conditions qualify:*
     - ***a. Structural/Layout/Relational Breakdown***: *The organization of elements or their spatial relationships have completely collapsed.*
       Example: Code intends to generate a geometric puzzle of ***neatly arranged*** *shapes, but the image shows them* ***randomly piled up with misaligned vertices****.*
     - ***b. Categorical Difference***: *The core scene or element type is entirely different.*
       Example: Code draws a bar chart, but the image is a scatter plot.
     - ***c. Severe Mismatch of Key Elements/Attributes***: *The* ***quantity****, type, or core attributes of key elements are severely missing or incorrect.*
       Example: Code intends to draw a 10-pointed star, but the image shows only 3 unrelated lines.
     - ***d. Structural Collapse or Loss of Integrity***: *The elements in the image are fragmented or incomplete, failing to form the complete structure intended by the code.*
       Example: Code intends to draw a ***complete, closed ellipse****, but the image only shows two* ***separate, unclosed curves****.*
     - ***e. Data-Level Misalignment***: *The data order, sorting, or mapping relationship between elements does not match the code's intent.*
       Example: Code intends to connect points in the order `A->B->C`*, but the image shows* `A->C->B`*.*
3. ***Instructions***
   Now, here is the code used to generate the image: {input code this}
   Strictly follow the process above, analyze step-by-step, provide reasons, and then generate the complete analysis report using the specified Markdown format below.
4. ***Output Format***

```
# Image Quality Analysis Report
## Consistency Review
* Verdict: [Enter ``Sufficient Match" or ``Fundamental Mismatch"]
* Reason: [Fill this in only if the verdict is ``Fundamental Mismatch", explaining
```

Image-Code Consistency Prompt $\mathcal{Q}_{\text{IC}}$ (continued)

```
    the type and specific phenomenon of the mismatch.]
```

B. Native Image-Code Pairs *v.s.* Explanatory Image-Code Pairs

1 The Definition of Native Image-Code Pairs

Native image-code pairs consist of raw visual image paired with ground-truth executable code without additional explanatory context. While these pairs ensure code correctness, they lack the pedagogical scaffolding necessary for models to learn the visual-to-code mappings and the rationale for specific implementation choices.

2 The Definition of Explanatory Image-Code Pairs

Explanatory image-code pairs augment ground-truth code with step-by-step descriptions, explicit parameter justifications, and instructional commentary that articulates how visual elements map to code constructs. This explanatory richness enables models to understand not merely what code to generate, but why particular implementations best capture the visual content.

To illustrate the enhanced instructional value of explanatory image-code pairs, we present a representative example below. The example demonstrates how the explanatory framework transforms bare code into comprehensive learning materials by integrating visual observation, implementation logic, and detailed commentary.

A Explanatory Image-Code Pair Example

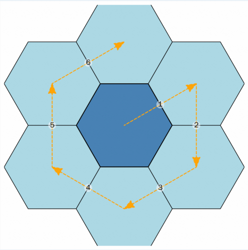


Figure 3. The hexagonal grid with a spiraling path, generated by the Python code below.

An analysis of the image reveals a honeycomb structure composed of seven regular hexagons. A central, darker blue hexagon is surrounded by six lighter blue hexagons. A dashed orange path originates inside the central hexagon and spirals outwards, passing through the vertices of the central hexagon. Below is a breakdown of the steps to reproduce this image using Python's matplotlib library, followed by the complete, executable code.

1. *Implementation Breakdown*

To reproduce this, we can use the `matplotlib` *library in Python. Here's a breakdown of the steps:*

1. ***Hexagon Grid**: We define a function to calculate the vertices of a regular "flat-top" hexagon given its center and size. The central hexagon is placed at the origin* $(0, 0)$*, and the six surrounding hexagons are arranged in a standard hexagonal grid pattern around it.*
2. ***Drawing Hexagons**: The* `matplotlib.patches.Polygon` *class is used to create the hexagon shapes. The central hexagon is colored* `steelblue`*, and the outer ones are* `lightblue`*. All have black edges, with the central one having a slightly thicker border.*
3. ***Defining the Path**: The path is a complex polyline. It starts near the center, moves to the first vertex (*p_1*), then "bounces" into the adjacent hexagon to an intermediate point (*mid_1*), and then connects to the next vertex (*p_2*). This pattern repeats for all six vertices of the central hexagon (*p_1 *through* p_6*).*
4. ***Drawing the Path and Arrows**: The path is constructed from two types of segments:*
   - *Simple dashed lines, drawn with* `ax.plot()`*.*
   - *Dashed lines with an arrowhead, drawn with* `ax.annotate()`*. The annotate function is used here to draw an arrow from a start point (*`xytext`*) to an end point (*`xy`*) with specific styling (dashed, orange, with a particular arrowhead shape). This is used for the first leg of each "bounce" (e.g.,* $p_1 \rightarrow mid_1$*).*
5. ***Labels and Markers**: The vertices 1 through 6 are marked with a small white circle with a black outline, created using* `ax.plot()` *with an* `'o'` *marker. The corresponding numbers are placed nearby using* `ax.text()`*. The positions of the*

A Explanatory Image-Code Pair Example (continued)

labels and the intermediate path points are carefully chosen to match the original image.

6. ***Final Touches**: The axes are turned off, and the plot limits are set to frame the hexagons similarly to the original image, creating a clean and focused visual.*

2. ***Python Implementation Code***

The following code reproduces the honeycomb structure:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.patches import Polygon

def generate_hexagon_path_image():
    """
    Generates the code to reproduce the image of
    a hexagonal grid with a path.
    """
    # 1. Setup figure and axes
    fig, ax = plt.subplots(figsize=(10, 10))
    ax.set_aspect(’equal’)
    ax.axis(’off’)
    fig.set_facecolor(’white’)

    # 2. Parameters
    s = 1.0  # Hexagon side length
    dark_blue_color = ’steelblue’
    light_blue_color = ’#add8e6’
    orange_color = ’orange’
    line_props = dict(color=orange_color,
                      linestyle=’--’,
                      linewidth=2,
                      zorder=2)
    arrow_props = dict(
        arrowstyle=’->, head_length=0.7, head_width=0.4’,
        **line_props)

    # 3. Hexagon generation function (flat-top)
    def get_flat_top_hexagon_vertices(center, size):
        x, y = center
        return np.array([
            [x + size, y],
            [x + size / 2, y + size * np.sqrt(3) / 2],
            [x - size / 2, y + size * np.sqrt(3) / 2],
            [x - size, y],
            [x - size / 2, y - size * np.sqrt(3) / 2],
            [x + size / 2, y - size * np.sqrt(3) / 2],
        ])

    # 4. Draw hexagons
    center_main = np.array([0, 0])
    vertices_main = get_flat_top_hexagon_vertices(
        center_main, s)
    ax.add_patch(Polygon(vertices_main,
                         facecolor=dark_blue_color,
                         edgecolor=’black’,
                         linewidth=2))

    centers_outer = [
        center_main + [0, s * np.sqrt(3)],
        center_main + [s * 1.5, s * np.sqrt(3) / 2],
        center_main + [s * 1.5, -s * np.sqrt(3) / 2],
        center_main + [0, -s * np.sqrt(3)],
        center_main + [-s * 1.5, -s * np.sqrt(3) / 2],
        center_main + [-s * 1.5, s * np.sqrt(3) / 2],
```

A Explanatory Image-Code Pair Example (continued)

```
    ]
    for center in centers_outer:
        ax.add_patch(Polygon(
            get_flat_top_hexagon_vertices(center, s),
            facecolor=light_blue_color,
            edgecolor=’black’,
            linewidth=1))

    # 5. Define path vertices and intermediate points
    p2, p1, p6, p5, p4, p3 = vertices_main

    mid1 = np.array([1.05, 0.45])
    mid2 = np.array([1.05, -0.45])
    mid3 = np.array([0, -1.3])
    mid4 = np.array([-1.05, -0.45])
    mid5 = np.array([-1.05, 0.45])
    mid6 = np.array([-0.2, 1.4])

    path_start = np.array([-0.4, 0])
    path_final_end = np.array([-0.7, 1.8])

    # 6. Draw the path segments
    ax.plot([path_start[0], p1[0]],
            [path_start[1], p1[1]],
            **line_props)

    path_segments = [
        (p1, mid1, p2),
        (p2, mid2, p3),
        (p3, mid3, p4),
        (p4, mid4, p5),
        (p5, mid5, p6)
    ]

    for start_node, mid_node, end_node in path_segments:
        ax.annotate(’’, xy=mid_node,
                    xytext=start_node,
                    arrowprops=arrow_props)
        ax.plot([mid_node[0], end_node[0]],
                [mid_node[1], end_node[1]],
                **line_props)

    ax.annotate(’’, xy=mid6, xytext=p6,
                arrowprops=arrow_props)
    ax.plot([mid6[0], path_final_end[0]],
            [mid6[1], path_final_end[1]],
            **line_props)

    # 7. Add vertex markers and labels
    vertex_map = {’1’: p1, ’2’: p2, ’3’: p3,
                  ’4’: p4, ’5’: p5, ’6’: p6}
    label_positions = {
        ’1’: (p1[0] - 0.25, p1[1] - 0.1),
        ’2’: (p2[0] + 0.15, p2[1] - 0.1),
        ’3’: (p3[0] + 0.15, p3[1] - 0.1),
        ’4’: (p4[0] - 0.25, p4[1] - 0.1),
        ’5’: (p5[0] - 0.2, p5[1]),
        ’6’: (p6[0] - 0.25, p6[1] - 0.1),
    }

    for num, pos in vertex_map.items():
        ax.plot(pos[0], pos[1], ’o’,
                markersize=6,
                markerfacecolor=’white’,
```

A Explanatory Image-Code Pair Example (continued)

```
                markeredgecolor='black',
                zorder=4)
        label_pos = label_positions[num]
        ax.text(label_pos[0], label_pos[1], num,
                fontsize=14,
                ha='center',
                va='center',
                zorder=5)

    # 8. Finalize plot
    ax.set_xlim(-2.8, 2.8)
    ax.set_ylim(-2.8, 2.8)

    plt.show()

if __name__ == '__main__':
    generate_hexagon_path_image()
```

3. ***Usage Instructions***
    - *Requires: matplotlib, numpy libraries*
    - *Run the script to generate the visualization*
    - *Adjust parameters in section 2 to customize appearance*
    - *Modify `s` variable to change hexagon size*

   Note: This structured approach allows for faithful reproduction of hexagonal grid patterns with customizable paths.

C. Native Caption *v.s.* Code-Grounded Caption

Comparisons

In this subsection, we present a representative example to illustrate the comparative advantages of our approach. The provided image poses a significant challenge: it contains an 8×8 grid with varying dot patterns in each cell, interconnected by red arrows that simulate the L-shaped movement of a knight in Chinese chess, traversing all cells to form a complex path.

We observe that even state-of-the-art MLLMs, such as Gemini 2.5 Pro, produce detailed descriptions riddled with factual errors when analyzing this image—misidentifying node counts (e.g., claiming 1 node instead of 2 in cell (1,8)), describing incorrect spatial arrangements, and reporting 68 arrows instead of the actual 63.

In contrast, our code-grounded captions deliver precise, verifiable information. By anchoring descriptions in executable code that can be validated against actual image data, this approach systematically eliminates perceptual hallucinations common in vision-language models, ensuring the blueprint-level accuracy essential for tasks demanding exact spatial reasoning and faithful reconstruction.

Comparisons on Native Caption and Code-Grounded Caption

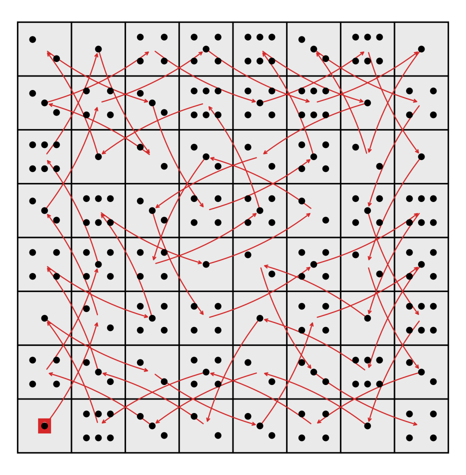


Figure 4. A representative image, where poses a significant challenge to the perceptual abilities of current MLLMs.

`Model Response:` *Of course. Here is a detailed description of the image, designed to empower AI models without visual capabilities to reason and provide human experts with blueprint-level reconstructions.*

1. ***Overall Layout and Style***
   This image is a square, minimalist abstract schematic diagram. Its core is a grid divided by black lines, with the grid cells filled in a light gray. Numerous solid black dots are scattered throughout the grid, a special starting point is marked by a red square, and there is a complex network of ~~numerous~~several red arrows connecting different ~~black dots~~cells. The overall image has no border, and the background is pure white.

2. ***Coordinate System***
   To precisely describe the position of all elements in the image, we will establish a Cartesian coordinate system based on the grid. The grid has an 8x8 structure. We will number the vertical columns from 1 to 8, moving from left to right, and the horizontal rows from 1 to 8, moving from bottom to top. Therefore, the cell in the bottom-left corner has the coordinates (1, 1), and the cell in the top-right corner has the coordinates (8, 8). When describing the black dots within each cell, we will use relative positional terms such as “center,” “top-left,” “bottom-right,” “top-center,” etc.

3. ***Grid and Background***
   The main body of the image is an 8x8 grid composed of thick black lines, containing a total of 64 perfectly identical square cells. The interior of each cell is filled with a light gray color (approximate hex code: #E0E0E0).

4. ***Detailed Element Description***

   - ***4.1 Black Dots (Nodes)***
     The image contains a total of ~~206~~216 uniformly sized, solid black dots (nodes). These nodes are distributed across the 64 grid cells, with the number and arrangement of nodes varying in each cell. One of these nodes is specially marked as the starting point. Below is a detailed distribution of the nodes in each cell, ordered from top to bottom (Row 8 to Row 1) and left to right (Column 1 to Column 8):
     Row 8 (Top)
     - ***Cell (1, 8):*** *~~1~~2 node, located at the ~~center.~~arranged diagonally, located at “top-left” and “bottom-right.”.*
     - ***Cell (2, 8):*** *~~3~~1 nodes, ~~arranged horizontally in the upper half of the cell. We will refer to them as “left,” “middle,” and “right.”~~located at the center.*
     - ***Cell (3, 8):*** *~~3~~4 nodes, ~~arranged horizontally in the upper half of the cell. We will refer to them as “left,” “middle,” and “right.”~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”*
     - ***Cell (4, 8):*** *~~3~~5 nodes, ~~arranged horizontally in the upper half of the cell. We will refer to them as “left,” “middle,” and “right.”~~arranged in a quincunx pattern, located at “top-left,” “top-right,” “center,” “bottom-left,” and “bottom-right.”*
     - ***Cell (5, 8):*** *~~2~~6 nodes, ~~arranged diagonally, located at “top-left” and “bottom-right.”~~arranged in two horizontal rows of three nodes each.*

Comparisons on Native Caption and Code-Grounded Caption (continued)

- **Cell (6, 8):** 3 nodes, ~~arranged horizontally in the upper half of the cell. We will refer to them as “left,” “middle,” and “right.”~~arranged diagonally, located at “top-left”, “center”, and “bottom-right.”
- **Cell (7, 8):** ~~3~~6 nodes, ~~arranged horizontally in the upper half of the cell. We will refer to them as “left,” “middle,” and “right.”~~arranged in two horizontal rows of three nodes each.
- **Cell (8, 8):** ~~3~~1 nodes, ~~arranged horizontally in the upper half of the cell. We will refer to them as “left,” “middle,” and “right.”~~located at the center.

Row 7

- **Cell (1, 7):** ~~1~~3 node, ~~located at the center.~~arranged diagonally, located at “top-left”, “center”, and “bottom-right.”
- **Cell (2, 7):** 4 nodes, arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”
- **Cell (3, 7):** 3 nodes, ~~forming a ’¡’ shape pointing left, located at “top-right,” “middle,” and “bottom-right.”~~arranged diagonally, located at “top-left”, “center”, and “bottom-right.”
- **Cell (4, 7):** ~~3~~6 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged in two horizontal rows of three nodes each.
- **Cell (5, 7):** ~~4~~5 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged in a quincunx pattern, located at “top-left,” “top-right,” “center,” “bottom-left,” and “bottom-right.”
- **Cell (6, 7):** ~~3~~6 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged in two horizontal rows of three nodes each.
- **Cell (7, 7):** ~~4~~1 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~located at the center.
- **Cell (8, 7):** ~~3~~4 nodes, ~~forming a ’¡’ shape pointing left, located at “top-right,” “middle,” and “bottom-right.”~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”

Row 6

- **Cell (1, 6):** ~~1~~6 node, ~~located at the center.~~arranged in two horizontal rows of three nodes each.
- **Cell (2, 6):** ~~3~~1 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~located at the center.
- **Cell (3, 6):** ~~4~~2 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left” and “bottom-right.”
- **Cell (4, 6):** ~~4~~3 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left”, “center”, and “bottom-right.”
- **Cell (5, 6):** ~~4~~2 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left” and “bottom-right.”
- **Cell (6, 6):** ~~3~~5 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged in a quincunx pattern, located at “top-left,” “top-right,” “center,” “bottom-left,” and “bottom-right.”
- **Cell (7, 6):** ~~3~~2 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left” and “bottom-right.”
- **Cell (8, 6):** ~~3~~1 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~located at the center.

Row 5

- **Cell (1, 5):** ~~1~~3 node, ~~located at the center.~~arranged diagonally, located at “top-left”, “center”, and “bottom-right.”
- **Cell (2, 5):** ~~4~~6 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged in two horizontal rows of three nodes each.
- **Cell (3, 5):** 3 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left”, “center”, and “bottom-right.”
- **Cell (4, 5):** 4 nodes, arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”
- **Cell (5, 5):** ~~4~~5 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged in a quincunx pattern, located at “top-left,” “top-right,” ”center,” “bottom-left,” and “bottom-right.”
- **Cell (6, 5):** ~~3~~2 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left” and “bottom-right.”
- **Cell (7, 5):** ~~4~~5 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged in a quincunx pattern, located at “top-left,” “top-right,” ”center,” “bottom-left,” and “bottom-right.”
- **Cell (8, 5):** ~~3~~6 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged in two horizontal rows of three nodes each.

Comparisons on Native Caption and Code-Grounded Caption (continued)

Row 4

- ***Cell (1, 4):*** *~~1~~4 node, ~~located at the center.~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”*
- ***Cell (2, 4):*** *~~3~~5 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged in a quincunx pattern, located at “top-left,” “top-right,” ”center,” “bottom-left,” and “bottom-right.”*
- ***Cell (3, 4):*** *4 nodes, arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”*
- ***Cell (4, 4):*** *~~4~~1 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~located at the center.*
- ***Cell (5, 4):*** *~~4~~2 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left” and “bottom-right.”*
- ***Cell (6, 4):*** *~~4~~5 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged in a quincunx pattern, located at “top-left,” “top-right,” ”center,” “bottom-left,” and “bottom-right.”*
- ***Cell (7, 4):*** *~~4~~2 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left” and “bottom-right.”*
- ***Cell (8, 4):*** *~~4~~5 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged in a quincunx pattern, located at “top-left,” “top-right,” ”center,” “bottom-left,” and “bottom-right.”*

Row 3

- ***Cell (1, 3):*** *1 node, located at the center.*
- ***Cell (2, 3):*** *~~4~~2 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left” and “bottom-right.”*
- ***Cell (3, 3):*** *~~3~~5 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged in a quincunx pattern, located at “top-left,” “top-right,” ”center,” “bottom-left,” and “bottom-right.”*
- ***Cell (4, 3):*** *4 nodes, arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”*
- ***Cell (5, 3):*** *~~4~~1 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~located at the center.*
- ***Cell (6, 3):*** *~~3~~4 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”*
- ***Cell (7, 3):*** *~~4~~1 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~located at the center.*
- ***Cell (8, 3):*** *~~3~~6 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged in two horizontal rows of three nodes each.*

Row 2

- ***Cell (1, 2):*** *~~1~~4 node, ~~located at the center.~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.” **~~This is the special starting point.~~**From (1, 1) to (2, 3)*
- ***Cell (2, 2):*** *3 nodes, ~~arranged in a pyramid shape, located at “top-center,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left”, “center”, and “bottom-right.”*
- ***Cell (3, 2):*** *~~4~~2 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left” and “bottom-right.”*
- ***Cell (4, 2):*** *~~4~~5 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged in a quincunx pattern, located at “top-left,” “top-right,” ”center,” “bottom-left,” and “bottom-right.”*
- ***Cell (5, 2):*** *~~4~~2 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left” and “bottom-right.”*
- ***Cell (6, 2):*** *~~4~~3 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left”, “center”, and “bottom-right.”*
- ***Cell (7, 2):*** *~~4~~6 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged in two horizontal rows of three nodes each.*
- ***Cell (8, 2):*** *~~4~~3 nodes, ~~arranged in a 2x2 square, located at “top-left,” “top-right,” “bottom-left,” and “bottom-right.”~~arranged diagonally, located at “top-left”, “center”, and “bottom-right.”*

Row 1 (Bottom)

- ***Cell (1, 1):*** *1 node, located at the center. **This is the special starting point.***
- ***Cell (2, 1):*** *~~3~~6 nodes, ~~arranged horizontally in the lower half of the cell. We will refer to them as “left,” “middle,” and “right.”~~arranged in two horizontal rows of three nodes each.*

Comparisons on Native Caption and Code-Grounded Caption (continued)

- **Cell (3, 1):** ~~4~~3 nodes, ~~arranged in a 2x2 square, located at "top-left," "top-right," "bottom-left," and "bottom-right."~~arranged diagonally, located at "top-left", "center", and "bottom-right."
- **Cell (4, 1):** ~~3~~2 nodes, ~~arranged horizontally in the lower half of the cell. We will refer to them as "left," "middle," and "right."~~arranged diagonally, located at "top-left" and "bottom-right."
- **Cell (5, 1):** ~~4~~3 nodes, ~~arranged in a 2x2 square, located at "top-left," "top-right," "bottom-left," and "bottom-right."~~arranged diagonally, located at "top-left", "center", and "bottom-right."
- **Cell (6, 1):** ~~3~~4 nodes, ~~arranged horizontally in the lower half of the cell. We will refer to them as "left," "middle," and "right."~~arranged in a 2x2 square, located at "top-left," "top-right," "bottom-left," and "bottom-right."
- **Cell (7, 1):** ~~4~~1 nodes, ~~arranged in a 2x2 square, located at "top-left," "top-right," "bottom-left," and "bottom-right."~~located at the center.
- **Cell (8, 1):** 4 nodes, arranged in a 2x2 square, located at "top-left," "top-right," "bottom-left," and "bottom-right."

- ***4.2 Starting Point***

  In cell (1, 1), the single central node is specially marked as the starting point. It is enclosed by a square slightly smaller than the cell itself. This square is filled with a bright red color (approximate hex code: #D82428) ~~and has a thick black border.~~.

- ***4.3 Red Arrows (Connections)***

  There are a total of ***~~68~~63*** *red arrows in the image. Each arrow is a straight line originating from the center of one black node and pointing to the center of another, with an arrowhead at its tip to indicate direction. These arrows crisscross to form a complex network. The following is a complete list of all ~~68~~63 arrows, describing their origin and destination points:*

  1. ~~From (1, 1, center) to (2, 3, top-left)~~From (1, 1) to (2, 3)
  2. ~~From (1, 2, center) (Start Point) to (2, 1, left)~~From (2, 3) to (1, 5)
  3. ~~From (1, 5, center) to (2, 4, top-center)~~From (1, 5) to (2, 7)
  4. ~~From (1, 6, center) to (2, 8, left)~~From (2, 7) to (4, 8)
  5. ~~From (1, 7, center) to (2, 7, bottom-right)~~From (4, 8) to (6, 7)
  6. ~~From (1, 8, center) to (1, 7, center)~~From (6, 7) to (8, 8)
  7. ~~From (2, 1, left) to (1, 1, center)~~From (8, 8) to (7, 6)
  8. ~~From (2, 2, bottom-left) to (3, 3, top-center)~~From (7, 6) to (6, 8)
  9. ~~From (2, 2, bottom-right) to (1, 3, center)~~From (6, 8) to (8, 7)
  10. ~~From (2, 3, top-left) to (3, 4, bottom-left)~~From (8, 7) to (7, 5)
  11. ~~From (2, 4, top-center) to (2, 6, top-center)~~From (7, 5) to (8, 3)
  12. ~~From (2, 4, bottom-right) to (3, 2, top-right)~~From (8, 3) to (7, 1)
  13. ~~From (2, 5, bottom-right) to (1, 6, center)~~From (7, 1) to (5, 2)
  14. ~~From (2, 6, top-center) to (1, 5, center)~~From (5, 2) to (3, 1)
  15. ~~From (2, 7, bottom-right) to (3, 7, middle)~~From (3, 1) to (1, 2)
  16. ~~From (2, 8, left) to (1, 8, center)~~From (1, 2) to (2, 4)
  17. ~~From (3, 1, top-right) to (4, 3, top-left)~~From (2, 4) to (1, 6)
  18. ~~From (3, 2, top-right) to (4, 1, right)~~From (1, 6) to (2, 8)
  19. ~~From (3, 2, bottom-right) to (2, 2, bottom-left)~~From (2, 8) to (3, 6)
  20. ~~From (3, 3, top-center) to (4, 3, bottom-right)~~From (3, 6) to (1, 7)
  21. ~~From (3, 4, bottom-left) to (4, 5, bottom-right)~~From (1, 7) to (3, 8)
  22. ~~From (3, 5, bottom-right) to (4, 7, top-left)~~From (3, 8) to (5, 7)
  23. ~~From (3, 6, left) to (2, 5, bottom-right)~~From (5, 7) to (7, 8)
  24. ~~From (3, 7, middle) to (4, 8, left)~~From (7, 8) to (8, 6)
  25. ~~From (3, 7, top-right) to (2, 6, bottom-right)~~From (8, 6) to (7, 4)
  26. ~~From (4, 1, left) to (3, 1, top-right)~~From (7, 4) to (8, 2)
  27. ~~From (4, 2, top-left) to (3, 1, bottom-left)~~From (8, 2) to (6, 1)
  28. ~~From (4, 3, top-left) to (5, 5, top-left)~~From (6, 1) to (4, 2)
  29. ~~From (4, 3, bottom-right) to (5, 4, bottom-right)~~From (4, 2) to (2, 1)
  30. ~~From (4, 4, top-left) to (5, 3, top-left)~~From (2, 1) to (1, 3)
  31. ~~From (4, 5, bottom-right) to (3, 6, left)~~From (1, 3) to (3, 2)

Comparisons on Native Caption and Code-Grounded Caption (continued)

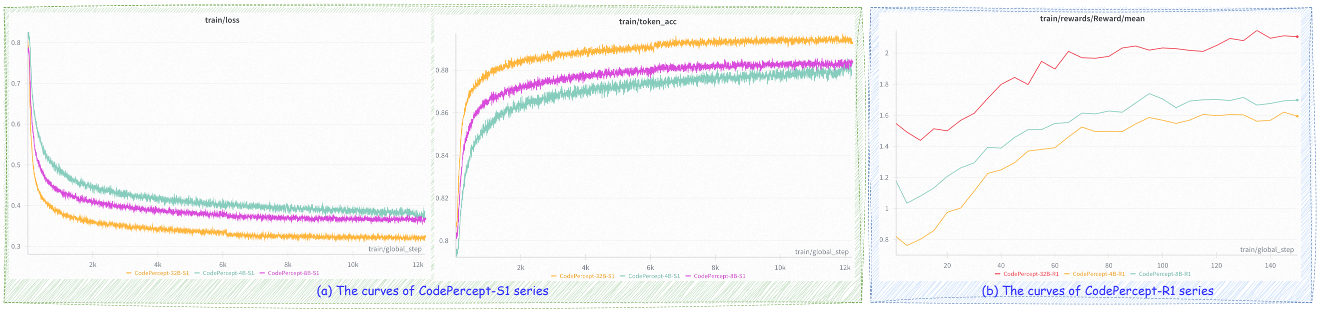
32. *~~From (4, 5, top-right) to (3, 5, top-center)~~From (3, 2) to (5, 1)*
33. *~~From (4, 6, bottom-left) to (3, 5, bottom-right)~~From (5, 1) to (6, 3)*
34. *~~From (4, 6, top-right) to (5, 8, bottom-right)~~From (6, 3) to (8, 4)*
35. *~~From (4, 7, top-left) to (5, 7, bottom-left)~~From (8, 4) to (7, 2)*
36. *~~From (4, 8, left) to (5, 8, top-left)~~From (7, 2) to (5, 3)*
37. *~~From (5, 1, middle-right) to (4, 2, top-left)~~From (5, 3) to (4, 1)*
38. *~~From (5, 2, bottom-left) to (6, 1, right)~~From (4, 1) to (2, 2)*
39. *~~From (5, 3, top-left) to (6, 3, bottom-right)~~From (2, 2) to (1, 4)*
40. *~~From (5, 4, bottom-right) to (4, 4, top-left)~~From (1, 4) to (3, 3)*
41. *~~From (5, 4, top-left) to (4, 4, top-right)~~From (3, 3) to (2, 5)*
42. *~~From (5, 5, bottom-right) to (5, 2, bottom-left)~~From (2, 5) to (4, 4)*
43. *~~From (5, 5, top-left) to (4, 6, bottom-left)~~From (4, 4) to (6, 5)*
44. *~~From (5, 6, top-right) to (4, 6, top-right)~~From (6, 5) to (4, 6)*
45. *~~From (5, 7, bottom-left) to (6, 7, top-center)~~From (4, 6) to (3, 4)*
46. *~~From (5, 8, top-left) to (6, 8, left)~~From (3, 4) to (5, 5)*
47. *~~From (6, 1, middle) to (5, 1, middle-right)~~From (5, 5) to (4, 7)*
48. *~~From (6, 2, bottom-left) to (5, 1, top-left)~~From (4, 7) to (2, 6)*
49. *~~From (6, 3, bottom-right) to (7, 3, top-right)~~From (2, 6) to (1, 8)*
50. *~~From (6, 4, top-right) to (7, 4, bottom-right)~~From (1, 8) to (3, 7)*
51. *~~From (6, 5, bottom-left) to (5, 6, top-right)~~From (3, 7) to (4, 5)*
52. *~~From (6, 6, top-right) to (5, 5, bottom-right)~~From (4, 5) to (6, 6)*
53. *~~From (6, 7, top-center) to (7, 7, bottom-right)~~From (6, 6) to (5, 8)*
54. *~~From (6, 8, left) to (7, 8, left)~~From (5, 8) to (7, 7)*
55. *~~From (7, 1, top-left) to (6, 2, bottom-left)~~From (7, 7) to (5, 6)*
56. *~~From (7, 2, bottom-right) to (8, 2, bottom-left)~~From (5, 6) to (3, 5)*
57. *~~From (7, 3, top-right) to (6, 5, bottom-left)~~From (3, 5) to (4, 3)*
58. *~~From (7, 4, bottom-right) to (8, 3, top-right)~~From (4, 3) to (6, 4)*
59. *~~From (7, 5, bottom-left) to (6, 4, top-right)~~From (6, 4) to (8, 5)*
60. *~~From (7, 6, bottom-left) to (8, 6, bottom-right)~~From (8, 5) to (7, 3)*
61. *~~From (7, 7, bottom-right) to (6, 6, top-right)~~From (7, 3) to (5, 4)*
62. *~~From (7, 8, left) to (8, 8, left)~~From (5, 4) to (6, 2)*
63. *~~From (8, 1, top-right) to (7, 1, top-left)~~From (6, 2) to (8, 1)*
64. *~~From (8, 2, bottom-left) to (8, 1, top-right)~~*
65. *~~From (8, 3, top-right) to (7, 2, bottom-right)~~*
66. *~~From (8, 5, bottom-left) to (7, 5, bottom-left)~~*
67. *~~From (8, 6, bottom-right) to (8, 5, bottom-left)~~*
68. *~~From (8, 7, bottom-right) to (7, 6, bottom-left)~~*

The above description provides the precise number, location, shape, attributes, and interrelationships of all visual elements in the image, which is sufficient to support in-depth analysis or accurate reproduction of the image.

D. Training Details

1 The Training Details of CodePercept-S1

CodePercept-S1 is built upon the Qwen3-VL-Instruct foundation model and fine-tuned using a carefully curated mixture of dataset ICC-1M, which is derived from public STEM seed data including Mulberry-SFT [57] and GRAMMAR [6]. The model is trained for 1 epochs using AdamW optimizer with DeepSpeed Zero-2 optimization, employing a cosine learning rate schedule starting from 3×10^{-6} and decaying to 2×10^{-7} with 5% warm-up steps. Training is conducted with an effective batch size of 4 per device (2 samples per device with gradient accumulation of 2 steps), using bfloat16 mixed precision and



(a) The curves of CodePercept-S1 series

(b) The curves of CodePercept-R1 series

Figure 5. The training curves of our proposed models. (a) the curves of CodePercept-S1 models. (b) the curves of CodePercept-R1 models.

Flash Attention for computational efficiency.

2 The Training Details of CodePercept-R1

CodePercept-R1 advances upon CodePercept-S1 through Group Relative Policy Optimization (GRPO), a reinforcement learning approach designed to enhance code generation quality. The model is initialized from the CodePercept-S1 checkpoint and further refined using a carefully selected subset of 10,000 high-quality samples from the ICC-1M dataset, chosen based on quality filter. We first perform a one-time difficulty filtering based on Qwen3-VL-7B: under 8 rollout iterations, we select samples whose accuracy falls between 0.25 and 0.75 [59]. We utilize GRPO with a temperature of 1.0, top-p sampling of 0.85, and a repetition penalty of 1.1 to encourage diverse and coherent generations. For each training sample, the model generates 8 candidate completions with a maximum length of 8,192 tokens, which are then optimized using a composite reward function. Training is conducted for 1 epoch using the AdamW optimizer with a learning rate of 1×10^{-6} and 5% warm-up steps. The optimization uses bfloat16 mixed precision and gradient clipping with a maximum norm of 0.5 for training stability. The GRPO algorithm uses a KL penalty coefficient (beta) of 0.001 to balance between reward optimization and maintaining proximity to the reference policy.

3 The Training Curves of CodePercept-S1 series and CodePercept-R1 series

As shown in Fig. 5, this image displays three charts illustrating the training performance of different versions of a model named “CodePercept”. The charts are divided into two groups: (a) for the “CodePercept-S1 series” and (b) for the “CodePercept-R1 series”. Both series compare models of three different sizes: 4 billion (4B), 8 billion (8B), and 32 billion (32B) parameters. Across both the S1 and R1 training stages, the charts consistently demonstrate a clear scaling law: larger models in the CodePercept family exhibit better performance. Whether in the supervised fine-tuning stage (S1) or the reinforcement learning stage (R1), the CodePercept model with a larger number of parameters (32B > 8B > 4B) demonstrates superior performance, namely lower loss, higher accuracy, and higher reward value.

E. The Construction Pipeline of STEM2Code-Eval Benchmark

In the section, we provide the specific construction pipeline of our STEM2Code-Eval benchmark, as shown in Fig E. The caption prompt is the same as `Native Caption Prompt` (see Sec.C), and the img+cap2code prompt is foremulated as follows:

Img+Cap2Code prompt

You are a top-tier programmer proficient in Python and Matplotlib.

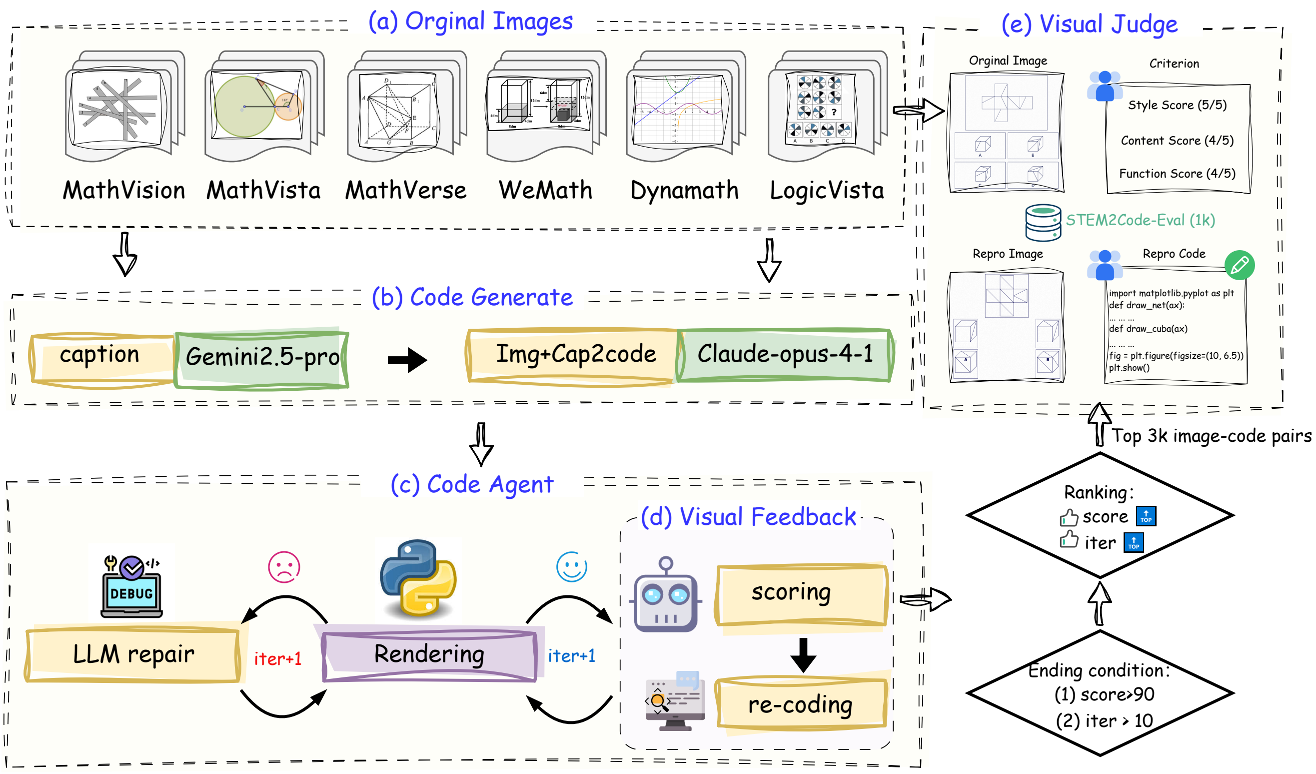
1. ***Primary Objective***

   Task*: Based on an input image and its corresponding detailed image description, generate a standalone, high-quality Python code to reproduce that image.*

   The precise description of the input image is:

   `[description]`

   a. ***Core Requirement****: Observe the image, and based on the* ***image description*** *you received and the extracted* ***core principles****, precisely reproduce the original image.*



```
import matplotlib.pyplot as plt
def draw_net(ax):
... ... ...
def draw_cuba(ax)

... ... ...
fig = plt.figure(figsize=(10, 6.5))
plt.show()
```

Figure 6. The pipeline of our proposed STEM2Code-Eval Benchmark, which includes 1000 image-code pairs that have been checked and revised by human annotators.

Img+Cap2Code prompt (continued)

b. ***Visual Matching***: *Must ensure complete consistency with the following aspects in the description:*
   - *Coordinate system range and scales*
   - *Colors, line widths, and styles*
   - *Element positions and sizes*
   - *If the description includes axes, ticks, grids, etc., they must be precisely reproduced.*

c. ***Code Algorithm***: *The code should not only generate the correct image, but also be written in a clear, structured manner that is intuitively reflected in the final image. The purpose is: An observer should be able to analyze the parameters or logic in the code by merely observing the rendered image, ultimately achieving Image2Code restoration.*

2. *Output Format & Prohibitions*
- *Your entire response must be a* ***single text block*** *with the following structure:*
  Python Code Block: *Strictly enclosed by* `'''python '''`.
- *Code is self-contained with import, one main function, and one main entry point.*
- *Code must include* `plt.show()` *at the end.*

Next, the LLM repair prompt is define as:

LLM Repair Prompt

You are a Code Debug Assistant. Your task is to identify and fix issues in the user's code based on any provided errors, ensuring it works correctly. You will be given the user's code and a corresponding error message in the following format:
Error code
`'''python [Error code] '''`

LLM Repair Prompt (continued)

Error message

`'''text [Error message] '''`

Your response must strictly adhere to the following criteria:

- *Return ONLY the complete, corrected Python code.*
- *The code must be enclosed within a single* `'''python... '''` *block.*
- *You must return all the complete, working code, not just the modified part.*

Next, the scoring prompt is defined as follows:

Image Scoring Prompt

You are an expert judge in evaluating mathematical and geometric diagrams. The first image (reference image) is a ground truth mathematical figure, and the second image (AI-generated image) is created using code generated by an AI assistant. Your task is to score how well the AI-generated image matches the ground truth image.

1. ***Scoring Methodology***

   Task*: The AI-generated image's score is based on the following criteria, totaling a score out of 100 points. The evaluation must consider the mathematical and geometric correctness of the figure, focusing on the precise arrangement and relationships of its components.*

   Criterion 1. Geometric & Structural Completeness (30 points)*:*
   - ***Element Types:*** *Does the AI-generated image include all fundamental element types from the reference image (e.g., points, lines, segments, rays, circles, polygons, curves, coordinate axes, text labels)?*
   - ***Element Quantity:*** *Is the* ***exact number*** *of each element type correct? (e.g., if the reference has 8 points and 3 triangles, does the generated image also have exactly 8 points and 3 triangles?).*

   Criterion 2. Positional & Relational Accuracy (30 points)*:*
   - ***Absolute & Relative Positioning:*** *Are all elements placed at their correct locations? This assesses accuracy within the image's implicit or explicit coordinate system.*
   - ***Spatial Relationships:*** *Does the image correctly represent all spatial relationships, such as* ***adjacency****,* ***intersection****,* ***containment****,* ***collinearity****, and* ***parallelism/perpendicularity****?*
   - ***Sequential & Topological Relationships:*** *For figures like graphs or paths, is the* ***sequence of connections*** *correct? Is the overall topological structure preserved?*
   - ***Layering (Z-order):*** *Are overlapping elements stacked in the correct order (e.g., is the shaded region correctly drawn behind the boundary line)?*

   Criterion 3. Text & Annotation Fidelity (10 points)*:*
   - ***Content:*** *Does the AI-generated image include all text and symbolic annotations from the reference? Is the content of the text identical?*
   - ***Positioning & Association:*** *Are annotations placed correctly relative to the geometric elements they describe?*
   - ***Style:*** *Does the style of the text (e.g., font, size, italics for variables, use of mathematical symbols) match the reference?*

   Criterion 4. Visual & Stylistic Consistency (20 points)*:*
   - ***Colors & Fill:*** *Do the colors (stroke, fill) of all elements match the reference? Are shaded regions filled correctly?*
   - ***Line & Marker Styles:*** *Do line styles (e.g., solid, dashed, dotted), line weights, and marker styles (e.g., dots, small circles, crosses) match?*
   - ***Overall Aesthetics:*** *Does the overall appearance, including background color, grid lines, and aspect ratio, match the reference image?*

   Criterion 5. Clarity & Legibility (10 points)*:*
   - *Is the AI-generated image clear, sharp, and well-rendered?*
   - *Are there any distracting artifacts, incorrect overlaps, or elements that are difficult to distinguish? Is all text legible?*

2. ***Evaluation & Response Format***
   - *Compare the two images head to head and provide a detailed assessment based on the criteria above.*

Image Scoring Prompt (continued)

- *Use the following format for your response to ensure the evaluation is clear and comprehensive.*

```
---
Comments:
- Geometric & Structural Completeness:  ${your comment and subscore}
- Positional & Relational Accuracy:  ${your comment and subscore}
- Text & Annotation Fidelity:  ${your comment and subscore}
- Visual & Stylistic Consistency:  ${your comment and subscore}
- Clarity & Legibility:  ${your comment and subscore}

Score:  ${your final score out of 100}
---
```

Please use the above format to ensure the evaluation is clear and comprehensive.

Next, the re-coding prompt is defined as:

Re-Scoring Prompt

You are an expert in Python data visualization, skilled at accurately recreating scientific charts through visual comparison.
You will receive three inputs:
`[Original Image]`*: The first image, our target for reproduction, originating from the fields of Science, Technology, Engineering, and Mathematics (STEM).*
`[Current Render]`*: The second image, a preliminary version generated by the code below.*
`[Current Code]`*: The Python code that generated the "Current Render".*
`'''python [Current Code] '''`.
Your core task is to identify the differences between the "Current Render" and the "Original Image" in terms of core scientific information and key structures, and then help me correct the code to reproduce the original image as closely as possible.

F. The Evaluation Implementation of STEM2Code-Eval Benchmark

In this section, we provide a detailed description of the implementation for the three metrics used in our STEM2Code-Eval benchmark: Image Score, Code Score, and Execution Rate (Exec Rate).

F.1. Image Score

The Image Score is designed to evaluate the visual fidelity and semantic correctness of the generated image compared to the ground-truth image. We leverage the advanced multi-modal understanding capabilities of Google's Gemini2.5-Pro for this task. The evaluation process is as follows:

1. **Input**: For each sample, we provide Gemini2.5 Pro with both the ground-truth image and the image generated by the model-under-test's code.
2. **Prompting**: We use a carefully crafted prompt that instructs the model to act as an expert evaluator. The prompt asks for a comprehensive comparison focusing on key visual elements, data representation accuracy, color schemes, layout, and overall similarity.
3. **Scoring**: The model is instructed to provide a final score on a scale of 0 to 100, where 0 indicates no resemblance and 100 indicates a pixel-perfect match. To ensure reliable parsing, we require the score to be formatted in a specific way (e.g., "Final Score: [score]").
4. **Extraction**: We programmatically parse the model's textual response to extract the numerical score for aggregation.

We used the same prompt for Gemini 2.5 Pro as described in Sec.E.

F.2. Code Score

The Code Score assesses the quality of the generated Python code itself. This evaluation goes beyond mere executability and focuses on correctness, readability, and best practices. We utilize OpenAI's GPT-4o for its strong code understanding and generation capabilities. The evaluation process is as follows:

1. **Input**: For each sample, we provide GPT-4o with the Python code generated by the model-under-test.
2. **Prompting**: The model is prompted to act as a senior software engineer conducting a code review. The prompt specifies multiple dimensions for evaluation: logical correctness, readability, efficiency, and robustness (e.g., handling of imports).
3. **Scoring**: The model is asked to provide a score for each dimension and then an overall quality score on a scale of 0 to 100. This multi-faceted approach provides a more nuanced assessment.
4. **Extraction**: The final overall score is extracted from the formatted response.

The specific prompt used for GPT-4o is as follows:

Code Scoring Prompt

You will act as an expert judge, responsible for rigorous visual verification of AI-generated graphics code. Your sole task is to evaluate whether the AI code is completely consistent with the reference code written by human experts in terms of the final rendered visual result. You must ignore technical differences in the code implementation (e.g., algorithms, data structures) and focus on every pixel and geometric detail that goes into rendering the final image.

1. ***Evaluation Mission and Core Principles***

   Criterion 1. *Visual Identity: Two pieces of code that render the exact same image should be considered equally valid. Elegance or clumsiness of the implementation is irrelevant to the scoring.*

   Criterion 2. *Pixel-Level Accuracy: Your evaluation must be accurate down to the pixel level. This includes geometric shape outlines, position, number of elements, relative relationships, and all visual attributes.*

   Criterion 3. *Objective and Quantitative: All comments must be supported by concrete visual evidence, strictly adhering to the following scoring criteria.*

   Criterion 4. *Unconditional Evaluation: The evaluation must be performed on any provided AI-generated code,* ***regardless of whether it is empty or incomplete****. You must be scored accordingly by applying the standard criteria, which will naturally result in a very low score.*

2. ***Scoring Criteria and Guidelines (100 points)***

   You will be scored based on the following five criteria. Each item is directly related to the final visual presentation.

   Criterion 1. *Overall Layout and Visual Attribute Fidelity (20 points)*
   - *Canvas and Coordinate System: Are the canvas attributes (e.g., aspect ratio, background color) correct? If a grid or coordinate system exists, are its range, scale, and scale consistent with the reference standard?*
   - *Macro Layout: Is the overall basic framework of the graphic correct? (For example, where is the main subject located on the canvas, and is the overall visual center of gravity consistent?)*
   - *Color and Style: Are the fill color, stroke color, and opacity of all elements consistent with the reference code? Do the line width, style (solid, dashed, dotted), and cap style (round, square) match?*
   - *Text and Annotations: If text labels or mathematical annotations exist, are their content, font, size, position, and alignment consistent with the reference code?*

   Criterion 2. *Quantitative Fidelity (20 points)*
   - *Element List Verification: Does the AI-generated graphic contain the exact same types and numbers of geometric elements as the reference code? (e.g., 8 polygons, 14 path nodes, 1 mesh).*
   - *Completeness: Are there any missing or redundant geometric components compared to the reference code?*

   Criterion 3. *Positioning and Layout Accuracy (30 points)*
   - *Absolute Coordinate Accuracy: Do the coordinates of all key elements (e.g., polygon vertices, circle centers, path anchor points) precisely match those calculated in the reference code?*
   - *Relative Position Relationship: Are the spatial arrangement of elements correct? (e.g., A is above and left of B, C and D are horizontally aligned, and a group of elements are arranged in a circular pattern).*
   - *Alignment and Distribution: Do the elements follow the same alignment (left/right/center) and*

Code Scoring Prompt (continued)

distribution (uniform/non-uniform) pattern as the reference code?

Criterion 4. *Relationship and Stacking Completeness (20 points)*
- *Connectivity and Sequence: If the graph contains paths, networks, or ordered sequences, is the order of connections between nodes **perfectly reproduced**? Are the starting and ending points of lines or paths correct?*
- *Spatial Interaction: Are complex relationships between elements (such as adjacency, containment, intersection, and overlap) rendered correctly? Are the shapes and sizes of overlapping areas accurate?*
- *Stacking Order (Z-index): When elements overlap, are they stacked in the correct order (i.e., which element is on top and which is on the bottom)?*

Criterion 5. *Code Implementation and Quality (10 points)*
- *Clarity and Readability: Is the code well-structured and clear? Does it use meaningful variable names and appropriate comments?*
- *Correctness and Efficiency: Is the code free of syntactical errors, logical errors, and unnecessary redundancy? Does it effectively use appropriate functions and methods from relevant libraries?*
- *Reproducibility: When executed in the correct environment, does the code run correctly and produce the expected complete graph? —*

3. *Evaluation*

Compare the reference code to the AI code. Provide a detailed evaluation and rating for each criterion, and then calculate a final overall score.

You must strictly adhere to the following format for your response. (Highest Priority)

```
---
Comments:
- Geometric & Structural Completeness: ${your comment and subscore}
- Positional & Relational Accuracy: ${your comment and subscore}
- Text & Annotation Fidelity: ${your comment and subscore}
- Visual & Stylistic Consistency: ${your comment and subscore}
- Clarity & Legibility: ${your comment and subscore}

Score: ${your final score out of 100}
---
```

Now, give your reference code and AI-generated code in the following format:

Reference Code

`'''python [Reference Code] '''`

AI-Generated Code

`'''python [AI-Generated Code] '''`

Please use the above format to ensure the evaluation is clear and comprehensive.

F.3. Exec Rate

The Execution Rate (Exec Rate) is a binary metric that measures the direct success rate of the generated code. It is the most objective of the three metrics. A code snippet is considered successful if and only if it executes without errors and produces the intended visual output file.

The automated execution pipeline is implemented as follows:

1. **Sandboxed Environment**: Each code snippet is executed in an isolated and clean sandboxed environment. This ensures that executions do not interfere with each other and provides a consistent environment. The environment is pre-configured with a standard set of Python data science libraries, including numpy, matplotlib, scipy, pandas, etc. To prevent rendering failures, necessary font libraries for handling Chinese characters and icons are also pre-installed.
2. **Execution**: The generated Python code is saved to a file (e.g., run.py) and executed using a Python interpreter. A timeout limit of 120 seconds is imposed to handle cases of infinite loops or excessively long computations.

3. **Verification**: Success is determined by a two-fold check:
    - The script must complete with an exit code of 0, indicating no runtime errors.
    - The script must generate an image file (e.g., .png, .jpg) in the working directory. The presence of this artifact is checked post-execution.
4. **Calculation**: The Exec Rate is calculated as the percentage of samples that pass both verification checks: Exec Rate $= \frac{\text{Number of Successfully Executed Samples}}{\text{Total Number of Samples}} \times 100$

G. Experiments with 32B

In this section, we continue to validate the effectiveness of our proposed CodePercept method in public six STEM benchmarks and our proposed STEM2Code-Eval benchmark. As shown in Tab.4 and Tab.5, CodePercept gets consistency improvements.

Table 4. Performance comparison of various MLLMs across six STEM reasoning benchmarks.

Image Captioner	Benchmark Datasets (%)						Average
	MathVision	MathVista	MathVerse	DynaMath	WeMath	LogicVista	
LLM Solver: Qwen3-30A3-Thinking [53]							
KeyeVL1.5-8B [42]	54.11	64.90	49.95	62.37	33.62	45.19	51.69
Intern-S1-8B [3]	51.67	65.70	51.90	63.61	33.43	51.23	52.92
GLM-4.1V-9B [20]	53.75	64.60	54.47	66.17	40.76	51.00	55.13
InternVL3.5-8B [48]	53.32	67.70	53.40	68.12	41.05	51.68	55.88
MiniCPM-V-4.5 [43]	53.15	66.60	57.84	65.44	43.71	52.57	56.55
Qwen2.5-VL-72B [4]	54.14	67.50	55.40	68.28	44.86	52.34	57.09
Qwen3-VL-30A3B-Instruct [36]	53.59	68.00	66.44	71.67	46.10	53.69	59.92
Claude-Opus 4.1-Thinking [1]	59.61	71.10	56.19	73.25	44.86	59.28	60.72
GPT5-Thinking [33]	60.03	65.20	69.56	71.00	54.57	53.02	62.23
Qwen3-VL-235A22B-Instruct	60.43	73.80	70.08	77.39	53.05	59.73	65.74
Gemini2.5-Pro	66.80	74.80	73.47	81.42	60.29	66.44	70.53
Qwen3-VL-4B-Instruct [36]	54.21	67.30	64.59	69.40	46.10	54.14	59.29
CodePercept-4B-S1	57.63**+3.4**	69.60**+2.3**	65.59**+1.0**	71.38**+2.0**	47.81**+1.7**	60.40**+6.3**	62.07**+2.8**
Qwen3-VL-8B-Instruct [36]	54.37	69.60	63.75	72.19	45.43	56.82	60.36
CodePercept-8B-S1	59.31**+5.0**	70.20**+0.6**	66.52**+2.8**	73.20**+1.0**	49.14**+3.7**	61.52**+4.7**	63.32**+3.0**
Qwen3-VL-32B-Instruct [36]	58.55	72.20	71.09	75.78	48.00	62.19	64.63
CodePercept-32B-S1	62.27**+3.7**	72.90**+0.7**	71.70**+0.6**	77.41**+1.6**	54.19**+6.2**	65.33**+3.1**	67.30**+2.7**
LLM Solver: Qwen3-235A22-Thinking [53]							
Qwen3-VL-4B-Instruct [36]	59.80	69.20	66.39	71.22	48.86	56.82	62.05
CodePercept-4B-S1	64.71**+4.9**	71.30**+2.1**	66.73**+0.3**	72.40**+1.2**	50.00**+1.1**	64.65**+7.8**	64.97**+2.9**
Qwen3-VL-8B-Instruct [36]	59.67	71.00	63.88	73.69	49.14	58.16	62.59
CodePercept-8B-S1	66.45**+6.8**	71.40**+0.4**	67.95**+4.1**	75.05**+1.4**	52.29**+3.2**	62.64**+4.5**	65.96**+3.4**
Qwen3-VL-32B-Instruct [36]	62.66	74.00	69.90	75.54	56.48	66.44	67.50
CodePercept-32B-S1	69.96**+7.3**	75.90**+1.9**	73.56**+3.6**	79.50**+4.0**	57.81**+1.3**	70.02**+3.6**	71.13**+3.6**

Table 5. Performance evaluation on our STEM2Code-Eval with 1k samples. We employ three metrics to assess their performances : (1) **Image Scoring** measures the visual similarity between the generated and original images; (2) **Code Scoring** assesses the quality, structure, and correctness of the generated Python code itself; (3) **Exec Rate** reports the execution success rate.

Model	**Image Score**	**Code Score**	**Exec Rate**	**Avg**
Intern-S1-8B [3]	6.02	15.87	26.60	16.16
InternVL3.5-8B [48]	13.50	18.16	56.50	29.38
MiniCPM-V-4.5 [43]	13.91	23.69	50.80	29.47
MiMo-VL-7B-RL [52]	14.54	22.41	60.30	32.42
Ovis2.5-9B [29]	9.76	11.26	89.40	36.81
KeyeVL1.5-8B [42]	20.33	22.47	73.40	38.73
GLM-4.1V-9B [20]	21.19	26.51	72.00	39.90
Qwen3-VL-4B-Thinking [36]	25.38	34.53	75.70	45.20
Qwen2.5-VL-72B-Instruct [4]	32.82	25.83	86.30	48.32
Qwen3-VL-8B-Thinking [36]	29.82	41.71	78.90	50.14
Qwen3-VL-30A3B-Instruct [36]	33.05	31.04	87.50	50.53
Seed1.6-Vision-nothinking [18]	31.22	38.56	85.50	51.76
Qwen3-VL-30A3B-Thinking [36]	37.47	35.53	87.10	53.37
Qwen3-VL-Plus-Instruct [36]	45.94	40.40	90.00	58.78
Qwen3-VL-Plus-Thinking [36]	45.59	40.61	89.20	58.47
Seed1.6-Vision-Thinking [18]	42.03	40.74	94.70	59.15
Gemini2.5-Flash-Thinking	57.25	60.87	85.20	67.77
Claude-Opus 4.1-Thinking	55.90	56.19	97.10	69.73
GPT5-Thinking	64.97	64.98	96.60	75.52
Gemini2.5-Pro-Thinking	68.89	75.41	91.70	78.67
Qwen3-VL-4B-Instruct [36]	24.55	26.42	79.40	43.46
CodePercept-4B-S1	38.13	43.43	80.70	54.09
CodePercept-4B-R1	**47.17**	**45.86**	91.30	**61.44**
Qwen3-VL-8B-Instruct [36]	28.59	28.23	85.30	47.37
CodePercept-8B-S1	44.53	46.78	87.60	**59.64**
CodePercept-8B-R1	**50.25**	**47.04**	93.40	**63.56**
Qwen3-VL-32B-Instruct [36]	36.85	39.98	81.80	52.88
CodePercept-32B-S1	61.14	56.99	93.00	**70.38**
CodePercept-32B-R1	**68.97**	**62.53**	**95.90**	**75.80**