A. Case study

1) **Query: What is the name of the landmark in the picture?**
Golden answer: Trondheimsfjorden
CoT step: *We attempt to analyze information about a landmark in an image, but I don't know what island it is.*
Relevance: True; Logic: True; Attribute: False.

Figure 7. Attribute errors due to external knowledge errors.

In this scenario, if GPT4V is utilized as the visual module for visual programming, it would accurately identify the landmark in the image as Munkholmen, a small islet located in the Trondheim Fjord in Norway. However, less capable models, such as blip2, would fail to provide the correct answer, subsequently leading to their detection by the SVIP system, which would mark the Attribute as false.

Failure case:

1) **Query: What is the length of the sides in the triangle below?**
Golden answer: 10
CoT step: *Finally, we leverage the relationship between the area of an isosceles right triangle and its side lengths to determine the length of one side. Given an area of 200, we rearrange the area formula to express the side length in terms of the area. By calculating the square root of twice the area, we find that each side measures 20.0 units, confirming the triangle's dimensions.*
Relevance: True; Logic: True; Attribute: True.

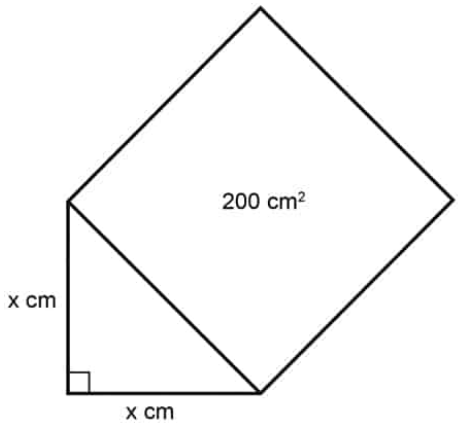

Figure 8. Enter Caption

In this example, the response exhibits an attribute error, as calculations should show that the sides of the triangle should be 10. However, SVIP failed to accurately identify the erroneous attribute of 20.0.

B. Details of training dataset

The details of the data mixture of image caption and VQA datasets used in MLLM training are shown in Table 7. We only use a subset of each dataset's training set. It should be noted that these data do not overlap with the data in Seedbench2. SVIP-Reward will provide step-level multi-dimensional reward signals for MLLM to complete these visual tasks during the reinforcement learning process.

Table 7. Data mixture of image caption and VQA datasets used in MLLM training.

Dataset	Description	Number of samples
COCO	Scene description	10.0K
Flickr 30K	Scene description	10.0K
VQAv2	General	100.0K
GQA	Compositional	86.0K
OK-VQA	Knowledge	9.0K
TallyQA	Counting	48.4K
Total		263.4K

C. Candidate sorting rules during inference time scaling

During the inference phase, it is essential to select the most appropriate candidate from N options. Given that the output of the SVIP-Reward is not a continuous score, we sort the three existing labels and establish the following selection criteria for choosing steps: When comparing two candidate steps, the one with a higher count of correct labels is prioritized. Subsequently, they are ranked based on the order of relevance, logic, and attributes as key indicators. For candidates that have identical labels across all three dimensions, an MLLM is utilized to discern the superior option. This process is particularly manageable given that there are only eight possible scenarios during reasoning. So we also show the ranking in Figure 9.

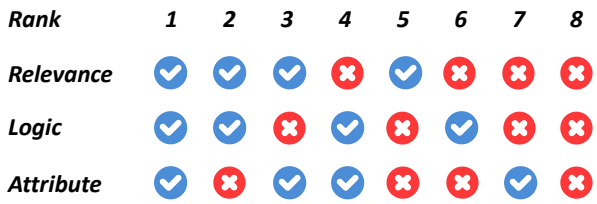

Figure 9. Candidate sorting rules.

During test-time scaling, there remains significant room for optimization in sampling and sorting, such as utilizing Monte Carlo Tree Search or Lookahead search strategies. Each method has its own corresponding and more efficient sorting approach. These aspects will be further explored in our dedicated research on test-time scaling.

D. Prompt used in the method.

We show the prompts used in the experiment as follows: For the given problem, generate the code for the first step:

Least-to-most code generation for the first step

Question: [QUESTION]
Notice: Determine the first step for implementing this question and output the corresponding code. In details, you MUST write a descriptive comment for the chosen first step firstly and then write the corresponding code for the first step. Ensure the comment format should be: '# Step 1: [Description of the step]'. Remember, you are required to provide only the descriptive comment and the corresponding codes for only the first step. Do not include any function definitions or return statement.

Least-to-most code generation for the following step

Question: [QUESTION]
Below are a few lines of code intended to solve the problem, which includes all the steps completed so far. If you think the existing code is sufficient to solve this question, make sure the code first prints the final result (which can be represented by a variable in the code) and then outputs 'Work is Done!'. Otherwise, you should determine the next appropriate step and its corresponding code based on the context of the existing code. In details, you only need to output a descriptive comment for your current chosen step and then write the corresponding codes for that step, and do not include any previous steps, function definitions or return statement. Ensure the comment format is: '# Step N: [Description of the step]', replacing N with the current step number.
[All the code steps completed so far]

Convert code snippets to corresponding step CoT

Task Description:
Please generate a single-step Chain-of-Thought (CoT) for the current step based on the provided code block and the values of the intermediate variables.
Requirements:
- The Chain-of-Thought should begin with "In this step, we use...".
- The explanation should not directly reference the variable names from the code block, but should include descriptions of the intermediate variables and its values.
- The Chain-of-Thought should be a concise, single-step explanation describing the logic of the code and how the intermediate variables support the reasoning.
- The description should be accurate, clear, and aligned with the programming logic.
Code Block: [CODE_BLOCK]
Intermediate Variables: [Values of intermediate variables]

PropTest's prompt for getting logic label

CONTEXT
The 'solve_query' function is a Python function that takes an image as input and implements the functionality described in [QUERY].
OBJECTIVE
Develop a Python function named 'execute_test' to verify whether the 'solve_query' function correctly implements the functionality described in [QUERY].
[EXAMPLES] are the in-context examples.
Include up to four test cases, each with the comment '# Test case n:' above the assert statement, starting from 1.
Consider these guidelines when creating the test cases:
1. Keep in mind that the return values do not contain numbers.
2. If the Query is True or False questions, the return values will be yes or no.
3. If the Query gives options using"or", the return values will be one of the options.
4. Use the llm_query function to answer informational questions not concerning the image.
STYLE
technical, in a correct Python format
TONE
clear, precise, professional
AUDIENCE
Developers and engineers who will use the test functions to verify the correctness of the solve_query function
RESPONSE
Provide the function that start with'def execute_test(image)' without any explanation.
Each test case should be commented with '#Test case n:' where 'n' represents the test case number.
Here are some [EXAMPLES]: [EXAMPLES]
Instruction
Generate the function execute_test for the following query:
[Query]: INSERT_QUERY_HERE

Define's prompt for getting attribute label

Visual Feedback (Image caption):
Please give me a short caption of [IMAGE2] located in the [BBOX] of [IMAGE1].
It mainly focuses on the connection between the object itself and the overall picture:
Visual Feedback (Sub-step verification):
[IMAGE_LIST] Is it possible that these pictures are the output of [substeps]?
Textual Feedback (Text summarization):
[TRACE] Please summarize in natural language what program trace outputs.
Textual Feedback (Logical verification):
[SUB_STEPS] Is it possible that these sub_steps can solve [QUERY]?
If not, please modify it and only start with #step1:, #step2 etc.:
Evaluator Prompt:
You are a program evaluator.
I will provide you with a piece of code and the feedback and results after it is executed.
Your task is to compare the target of the code and its result after calling the external function.
[Query]: [QUERY]
[Code]: [ORIGIN_CODE]
[Intermediate Variables]: [Values of intermediate variables]
[Visual Feedback]: [FEEDBACK_V]
[Textal Feedback]: [FEEDBACK_T]
[Compile Feedback]: [FEEDBACK_C]
Evaluate whether the output of the current code is correct: