

## A PSEUDOCODE DIAGRAM

We show a pseudocode algorithm here so that readers can better understand our process in Figure 2.

---

### Algorithm 1: The algorithm of **De-fine**

---

**Input:** An image  $x$  and a query  $q$   
**Output:** The *result* of the query  $q$  about image  $x$

- 1 Logic\_step = LLM\_decompose( $q$ );
- 2 Abstract\_code = sketch( $c$  in *codebase* if  $c.comments()$  relevant to  $q$ );
- 3 AL = Logic\_step, Abstract\_code;
- 4 **for**  $i$  in iteration **do**
- 5     Origin\_code = Code\_generator(API, AL,  $q$ );
- 6     Feedback = Feedback\_generator(Compiler(Origin\_code,  $x$ ));
- 7     Code = Refiner(Origin\_code, Feedback);
- 8 **end**
- 9 result = Compiler(Code,  $x$ )

---

## B THE IMPACT OF CAPTION AND INFORMATION

The improvements in performance are not solely due to providing captions or information for images. In the appendix, we conducted additional ablation experiments on the backbone, where the backbone was only given the captions of images and the textual traces after code execution. Comparing this data to our results in Table 5, there remains a noticeable difference. This indicates that it is our feedback mechanism, rather than the addition of information, that contributes to the performance improvements.

Table 8: Evaluation of Caption and Information.

	GQA
Backbone	52.6
+ caption	53.7
+ execution trace	52.8
+ compiler output	52.6

## C HUMAN EVALUATION

In this section, we explain the sources of error details of our experiments in Figure 5. We manually selected 100 samples from the GQA dataset, analyzed each program’s errors, and classified them into five categories: 1) Incorrect Program: For those programs that cannot be compiled, we all classify error codes. 2) Multiple Answer: As the function ‘best\_text\_match’ will return the most relevant string of image patch, there may be multiple correct results in the result. Therefore, the errors corresponding to this function are classified as Multiple Answer. 3) VQA: In the method, we have the ‘simple\_query’ function, the ‘llm\_query’ function and the ‘verify\_property’ function. These question-and-answer functions correspond to the Vqa function of Visual Programming. The program errors caused by these functions will be classified as VQA. 4) Location: Positioning is very critical in visual tasks. It is implemented by the ‘Loc’ function in Visual Programming and the ‘find’ function in **De-fine**. 5) Crop: In Visual Programming, there is the ‘Crop’ function, while in **De-fine** it is defined in the ‘ImagePatch’ class. Therefore, incorrect answers due to image cropping will be recorded here.

## D PROMPT USED IN THE MODEL.

We used the following prompt during our experiments:

### Logical Step Decompose:

```

1 Given a VQA query, output the decomposed sub-step to how to answer it.
2 The output must be brief and short.
3 Query: <QUERY>
4 Your answer should only start with #step1:, #step2 etc.:

```

### Code Generation:

```

1 You are a helpful assistant.
2 <API>
3 <AL_PROMPT>
4 <QUERY>
5 Only answer with a function starting def execute_command(image):

```

### Visual Feedback (Image caption):

```

1 Please give me a short caption of <IMAGE2> located in the <BBOX> of <IMAGE1>.
2 It mainly focuses on the connection between the object itself and the overall picture:

```

**Visual Feedback (Sub-step verification):**

```
1 <IMAGE_LIST> Is it possible that these pictures are the output of <substeps>?
```

**Textual Feedback (Text summarization):**

```
1 <TRACE> Please summarize in natural language what program trace outputs.
```

**Textual Feedback (Logical verification):**

```
1 <SUB_STEPS> Is it possible that these sub_steps can solve <QUERY>?
2 If not, please modify it and only start with #step1:, #step2 etc.:
```

**Refiner Prompt:**

```
1 You are a program optimizer (or corrector).
2
3 I will give you some information about how to use an API and a query about a picture.
4 Your task is to refine (or correct) the code that can answer the query by using API.
5 Please refine (or correct) the origin code based on feedbacks.
6
7 [API manual]: <API_MANUAL>
8 [Query]: <QUERY>
9 [Code]: <ORIGIN_CODE>
10 [Visual Feedback]: <FEEDBACK_V>
11 [Textal Feedback]: <FEEDBACK_T>
12 [Compile Feedback]: <FEEDBACK_C>
13 Your answer should start with:
14 def execute_command(image):
```

**E ERROR CASES**

1) **On the GQA task**, for the image named n172618.jpg, with the question “What is the height of the girl that is holding the kite?”, De-fine generated the following code:



**Figure 7: Image n172618.jpg with query “What is the height of the girl that is holding the kite?”**

```
1 def execute_command(image):
2     #Step1: Find the patch containing the girl
3     girl_patches = ImagePatch(image).find("girl")
4     #Step2: Find the girl holding the kite
5     kite_patches = ImagePatch(image).find("kite")
6     kite_patch = kite_patches[0]
7     girl_holding_kite_patches = []
8     for girl_patch in girl_patches:
9         if girl_patch.overlaps_with(kite_patch.lower, kite_patch.right, kite_patch.upper):
10             girl_holding_kite_patches.append(girl_patch)
11     #Step3: Calculate the height of the girl holding the kite
12     girl_holding_kite_patch = girl_holding_kite_patches[0]
13     height = girl_holding_kite_patch.upper - girl_holding_kite_patch.lower
14     return str(height) + "pixels"
```

**Listing 1: Code for image n172618.jpg**

Although the code is executed correctly and the logic was sound, the result was incorrect when compared to the expected answer “Tall”. This is because the question could not be adequately transformed into a classification problem of tall or short.

2) **On the OK-VQA task**, regarding the image COCO\_val2014\_000000029393.jpg with the question, "What type of dog is this?", De-fine produced the following code:



Figure 8: Image COCO\_val2014\_000000029393.jpg with query "What type of dog is this?"

```
1 def execute_command(image):
2     #Step1: Create an ImagePatch object from the image
3     image_patch = ImagePatch(image)
4     #Step2: Use simple_query method to get the answer
5     dog_breed = image_patch.simple_query("What type of dog is this?")
6     #Step3: Return the answer
7     return dog_breed
```

Listing 2: Code for image COCO\_val2014\_000000029393.jpg

The error in this instance was related to the use of the Blip tool in the `simple_query` function, which returned the answer "a dachshund". However, when we replaced the function called by `simple_query` with GPT-4V, "Rhodesian Ridgeback" was obtained. Although the label is "hound".

## F FEEDBACK GENERATOR OUTPUT EXAMPLE

Our feedback is multifaceted. We will showcase some examples from the experimental process and include them in the appendix:

1) **On the GQA task**, for the image named n172618.jpg, with the question "What is the height of the girl that is holding the kite?", De-fine generated the following feedback:

**Visual Feedback:** The young girl on the right in vibrant outdoor clothing holds onto a colorful kite, preparing to catch the wind on a lively day in the field. The young adventurer on the left in a pink sweater encounters the wonders of nature and the images you've provided do seem like they could be the result of an operation that extracts a specific area or "patch" from a larger image

**Textual Feedback:** There is a kite. There is a child holding the kite in her hand. The height of the child holding the kite is 355 pixels. Those sub-steps could be part of a process to estimate the height of the girl holding the kite.

**Compile Feedback:** Execution succeed!

2) **On the OK-VQA task**, regarding the image COCO\_val2014\_000000029393.jpg with the question, "What type of dog is this?", De-fine generated the following feedback:

**Visual Feedback:** A vigilant dog stands guard on a sunlit garden deck, with a lemon tree backdrop. From the context provided, it seems like these images could indeed be part of such a picture.

**Textual Feedback:** The breed of the dog is dachshund. But those steps alone are insufficient to determine the breed of a dog from an image. A specialized image recognition model would be required.

**Compile Feedback:** Execution succeed!

## G CODE GENERATED BY DIFFERENT ENGINES

In this section, we show the program output by different code generation engines and try to analyze them. Specifically, we take the same prompt as input and generate the corresponding codes by Code-Llama (34B), Code-Llama (70B), and GPT-3.5-Turbo (1106) respectively. In the subsequent exposition, we present a trio of illustrative examples involving multi-step queries and their corresponding responses, wherein the generation of code was efficaciously accomplished notwithstanding the limitations inherent in the model. Nevertheless, it is pertinent to acknowledge that the Code-Llama (70B) model exhibits occasional discrepancies in its interpretation of the 'verify\_property'

function. Concurrently, the Code-Llama (34B) variant demonstrates a partial deficiency in its capacity to decompose the problem at hand, potentially leading to lapses in logical reasoning.

**Query: Does the flag that is to the left of the bucket have large size and blue color?**

```

1 def execute_command(image):
2     #Step1: Find the flag and bucket patches
3     flag_patches = ImagePatch(image).find("flag")
4     bucket_patches = ImagePatch(image).find("bucket")
5     #Step2: Check if there is a flag to the left of the bucket
6     for flag_patch in flag_patches:
7         if flag_patch.horizontal_center < bucket_patches[0].horizontal_center:
8             #Step3: Check if the flag has large size and blue color
9             if flag_patch.verify_property("flag", "large") and flag_patch.verify_property("flag", "blue"):
10                return "yes"
11    #Step4: If no flag to the left of the bucket has large size and blue color, return "no"
12    return "no"

```

**Listing 3: Python program generated by GPT-3.5-Turbo (1106)**

```

1 def execute_command(image):
2     #Step1: Find the bucket and flag patches
3     image_patch = ImagePatch(image)
4     bucket_patches = image_patch.find("bucket")
5     flag_patches = image_patch.find("flag")
6     #Step2: Check if there is a flag to the left of the bucket
7     flag_patches_left = []
8     for flag_patch in flag_patches:
9         for bucket_patch in bucket_patches:
10            #Step3: Check if the flag has large size and blue color
11            if flag_patch.left > bucket_patch.left and flag_patch.right < bucket_patch.right:
12                flag_patches_left.append(flag_patch)
13    if len(flag_patches_left) == 0:
14        return "no"
15    #Step4: Return if flag to the left of the bucket has large size and blue color
16    flag_patch = flag_patches_left[0]
17    return bool_to_yn(flag_patch.verify_property("flag", "size", "large") and flag_patch.verify_property("flag", "color", "blue"))

```

**Listing 4: Python program generated by Code-Llama (70B)**

```

1 def execute_command(image):
2     #Step1: Find the flag and bucket patches
3     image_patch = ImagePatch(image)
4     flag_patches = image_patch.find("flag")
5     bucket_patches = image_patch.find("bucket")
6     flag_patch = flag_patches[0]
7     bucket_patch = bucket_patches[0]
8     #Step2: Return if flag to the left of the bucket has large size and blue color
9     return bool_to_yn(flag_patch.verify_property("flag", "large") and flag_patch.verify_property("flag", "blue") and flag_patch.verify_property("flag", "left", bucket_patch))

```

**Listing 5: Python program generated by Code-Llama (34B)**

**Query: Of what race is the man the skateboard is to the right of?**

```

1 def execute_command(image):
2     # Step1: Find the man patch
3     image_patch = ImagePatch(image)
4     man_patches = image_patch.find("man")
5     man_patch = man_patches[0]
6     # Step2: Find the skateboard patch to the right of the man
7     skateboard_patches = image_patch.find("skateboard")
8     right_skateboard_patches = []
9     for skateboard_patch in skateboard_patches:
10        if skateboard_patch.left > man_patch.right:
11            right_skateboard_patches.append(skateboard_patch)
12    # Step3: Find the race of the man the skateboard is to the right of
13    race = ""
14    if len(right_skateboard_patches) > 0:
15        race = right_skateboard_patches[0].simple_query("What is the race of the man?")
16    return race

```

**Listing 6: Python program generated by GPT-3.5-Turbo (1106)**

```

1509 1 def execute_command(image):
1510 2     # Step1: Find the skateboard patch
1511 3     image_patch = ImagePatch(image)
1512 4     skateboard_patches = image_patch.find("skateboard")
1513 5     skateboard_patch = skateboard_patches[0]
1514 6     # Step2: Find the man the skateboard is to the right of
1515 7     man_patches = image_patch.find("man")
1516 8     for man_patch in man_patches:
1517 9         if man_patch.right > skateboard_patch.right:
1518 10             # Step3: Return the race of the man
1519 11             return man_patch.simple_query("What race is this man?")

```

Listing 7: Python program generated by Code-Llama (70B)

```

1520 1 def execute_command(image):
1521 2     # Step1: Find the skateboard patch
1522 3     image_patch = ImagePatch(image)
1523 4     skateboard_patches = image_patch.find("skateboard")
1524 5     skateboard_patch = skateboard_patches[0]
1525 6     # Step2: Find the man patch
1526 7     man_patches = image_patch.find("man")
1527 8     man_patch = man_patches[0]
1528 9     # Step3: Return the race of the man the skateboard is to the right of
1529 10     if skateboard_patch.horizontal_center > man_patch.horizontal_center:
1530 11         return man_patch.simple_query("What is the race of the man?")
1531 12     else:
1532 13         return "No man in the image"

```

Listing 8: Python program generated by Code-Llama (34B)

**Query: Is the material of the computer monitor the same as the TV?**

```

1533 1 def execute_command(image):
1534 2     # Step1: Find the patches for the computer monitor and the TV
1535 3     image_patch = ImagePatch(image)
1536 4     monitor_patches = image_patch.find("computer monitor")
1537 5     tv_patches = image_patch.find("TV")
1538 6     # Step2: Check if the material of the computer monitor is the same as the TV
1539 7     monitor_material = monitor_patches[0].simple_query("What is the material?")
1540 8     tv_material = tv_patches[0].simple_query("What is the material?")
1541 9     is_same_material = monitor_material == tv_material
1542 10     # Step3: Return the answer
1543 11     return bool_to_yn(is_same_material)

```

Listing 9: Python program generated by GPT-3.5-Turbo (1106)

```

1544 1 def execute_command(image):
1545 2     # Step1: Find the computer monitor patches
1546 3     image_patch = ImagePatch(image)
1547 4     computer_patches = image_patch.find("computer monitor")
1548 5     # Step2: Check if the material of the computer monitor is the same as the TV
1549 6     tv_patches = image_patch.find("tv")
1550 7     for computer_patch in computer_patches:
1551 8         for tv_patch in tv_patches:
1552 9             if computer_patch.overlaps_with(tv_patch):
1553 10                 # Step3: Return the answer
1554 11                 return bool_to_yn(computer_patch.verify_property(tv_patch))

```

Listing 10: Python program generated by Code-Llama (70B)

```

1555 1 def execute_command(image):
1556 2     # Step1: Find the computer monitor and TV patches
1557 3     image_patch = ImagePatch(image)
1558 4     computer_monitor_patches = image_patch.find("computer monitor")
1559 5     tv_patches = image_patch.find("TV")
1560 6     # Step2: Check if the material of the TV is the same as the computer monitor
1561 7     tv_patch = tv_patches[0]
1562 8     computer_monitor_patch = computer_monitor_patches[0]
1563 9     return bool_to_yn(tv_patch.verify_property("TV", "material") == computer_monitor_patch.verify_property("computer monitor", "material"))

```

Listing 11: Python program generated by Code-Llama (34B)