CLOVA: A Closed-LOop Visual Assistant with Tool Usage and Update

Supplementary Material

1. Framework of CLOVA

The pseudo-code of CLOVA is summarized in Algorithm 1.

Algorithm 1 CLOVA

Input: LLMs, visual tools, instruction data $\mathcal{T} = \{T_1, T_2, \cdots, T_t\}$, demonstration pool $\mathcal{D}$, prompt pool $\mathcal{P} = \emptyset$.
Output: Updated $\mathcal{D}$, updated $\mathcal{P}$

for $i = 1, 2, \ldots, t$ **do**
  Perform inference for T_i by generating plan and program.
  if T_i is correctly solved **then**
    Save the plan and program to $\mathcal{D}$.
  else
    Convert intermediate results into language.
    Perform global reflection.
    Perform training-validation prompt tuning, store in-context examples into $\mathcal{D}$ and prompts in $\mathcal{P}$.
    if Updated tools solve T_i incorrectly **then**
      Perform local reflection.
      Perform training-validation prompt tuning, store in-context examples into $\mathcal{D}$ and prompts in $\mathcal{P}$.
    end if
  end if
end for

2. Comparisons with related methods

In Tab. 1, we present a more detailed comparison table with more related methods. In the table, 'Global' means the global reflection, 'Local' means the local reflection, 'Instruction' means instruction-following tuning, 'RL' means reinforcement learning, and 'Prompt' means using prompt examples as in-context learning. We observe that many tool-usage methods do not have a reflection capability, and few methods use global reflection to improve the plans or programs generated by LLMs. Different from them, CLOVA uses both global reflection and local reflection to identify tools that need to be updated, capable of handling complex instructions. Moreover, as we all know, our method is the first work to update visual tools, through which the visual assistant can better adapt to new environments.

3. Prompt Examples

3.1. In-context examples

In the inference phase of our method, CLOVA generates plans and programs based on in-context examples that include correct examples and incorrect examples with criteria. Here we show some correct examples and incorrect examples for the compositional VQA, multi-image reasoning, image editing, and factual knowledge tagging tasks, as shown in Figs. 1 to 4.

3.2. Prompts in inference

In the inference phase, we use LLMs to generate plans and programs. We show examples of prompts for plan generation and program generation in Figs. 5 and 6, respectively.

3.3. Prompts in reflection

In the reflection phase, we use LLMs for global reflection and local reflection. We show two examples of prompts for global reflection and local reflection in Figs. 7 and 8, respectively.

3.4. Prompts in learning

In the learning phase, we use LLMs to infer answers for the VQA tool, and then tune prompts of the VQA tool using the question and inferred answer. One example of prompts sent to LLMs for answer inferring is shown in Fig. 9, respectively.

4. Details of Tool update

4.1. Update VQA tool

4.1.1 Model

We use the BLIP [11] model for the VQA tool. One BLIP model is composed of three components: an image encoder, an image-grounded question encoder, and an answer coder, which are used to extract image features, extract question features, and generate answers, respectively. In our prompt tuning scheme, the visual features are extracted from the image encoder followed by average pooling, whose dimension is 768. We concatenate learnable prompts with the inputs to the answer decoder, guiding the answer decoder to generate desirable answers. The architecture of BLIP is shown in Fig. 10.

4.1.2 Training

The size of learnable prompts is $\mathbb{R}^{16\times768}$, that is we learn 16 vectors as prompts, and the dimension of each one is 768. Since it is non-trivial to define visual concepts for the VQA tool, we do not store concepts in the prompt pool of the VQA tool, and all learned prompts and stored together.

(a) Correct examples for plan generation

Instruction: Is the vehicle in the top of the image?
Plan:
Step1, Locate the upper region of the given image, and obtain bounding boxes of the upper region.
Step2, Crop the upper region from the given image, based on bounding boxes of the upper region. The bounding boxes are obtained in Step1.
Step3, Locate vehicle in the upper region of the given image, and obtain bounding boxes of vehicle. The upper region is cropped in Step2.
Step4, Count the number of vehicle, based on bounding boxes of vehicle. The bounding boxes are obtained in Step3.
Step5, Obtain whether the answer is 'yes' or 'no', based on the number of vehicles. The number is obtained in Step4. If the number is greater than zero, the answer is 'yes'; On the contrary, the answer is 'no'.
Step6, Visualize results.

(b) Incorrect examples for plan generation

Instruction: On which side of the picture are the yellow chairs, the right or the left?
Plan:
Step1, Locate the yellow chairs in the given image and obtain bounding boxes of the yellow chairs.
Step2, Crop the region where the yellow chairs are located from the given image, based on the bounding boxes of the yellow chairs. The bounding boxes are obtained in Step1.
Step3, Obtain whether the yellow chairs are on the right or left side of the picture, based on the position of the cropped region. If the region is on the right side of the picture, the answer is 'right'; On the contrary, the answer is 'left'.
Step4, Visualize the results.
Reason: The Step1-Step3 in the plan are correct. Using the position of the cropped region is not supported. It should first crop the left region of the image, and then detect where there are yellow chairs. If not, the answer is right.

(c) Correct examples for program generation

Instruction : Is the vehicle in the top of the image?
Plan:
Step1, Locate the upper region of the given image, and obtain bounding boxes of the upper region.
Step2, Crop the upper region from the given image, based on bounding boxes of the upper region. The bounding boxes are obtained in Step1.
Step3, Locate vehicle in the upper region of the given image, and obtain bounding boxes of vehicle. The upper region is cropped in Step2.
Step4, Count the number of vehicle, based on bounding boxes of vehicle. The bounding boxes are obtained in Step3.
Step5, Obtain whether the answer is 'yes' or 'no', based on the number of vehicles. The number is obtained in Step4. If the number is greater than zero, the answer is 'yes'; On the contrary, the answer is 'no'.
Step6, Visualize results.
Program:
```
BOX0=LOC(image=IMAGE,object='TOP')
IMAGE0=CROP(image=IMAGE,box=BOX0)
BOX1=LOC(image=IMAGE0,object='vehicle')
ANSWER0=COUNT(box=BOX1)
ANSWER1=EVAL(expr="'yes' if {ANSWER0} > 0 else 'no'")
FINAL_RESULT=RESULT(var=ANSWER1)
```

(d) Incorrect examples for program generation

Instruction : Is the lamp different in color than the shirt?
Plan:
Step1, Locate the lamp in the given image, and obtain bounding boxes of lamp.
Step2, Crop the region of the lamp from the given image, based on bounding boxes of lamp. The bounding boxes are obtained in Step1.
Step3, Asking the image region of lamp, 'What color is the lamp?'. The image region of lamp is cropped in Step2.
Step4, Locate the shirt in the given image, and obtain bounding boxes of shirt.
Step5, Crop the region of the shirt from the given image, based on bounding boxes of shirt. The bounding boxes are obtained in Step4.
Step6, Asking the image region of shirt, 'What color is the shirt?'. The image region of lamp is cropped in Step5.
Step7, Obtain whether the answer is 'yes' or 'no', based on the color of lamp and the color of shirt. The color of lamp and shirt is obtained in Step3 and Step6, respectively. If their color are the same, the answer is 'yes'; On the contrary, the answer is 'no'.
Step8, Visualize results.
Program:
```
BOX0=LOC(image=IMAGE,object='lamp')
IMAGE0=CROP(image=IMAGE,box=BOX0)
BOX1=LOC(image=IMAGE0,object='shirt')
ANSWER0=COUNT(box=BOX1)
ANSWER1=EVAL(expr="'yes' if {ANSWER0} > 0 else 'no'")
FINAL_RESULT=RESULT(var=ANSWER1)
```
Reason: The plan is correct, and can address the task. But the Step3-Step6 of the program does not match the plan. The plan locates, crops, and asks color of the lamp and shirt, but the program counts the number of lamp.

Figure 1. Demonstration examples for compositional VQA tasks in $\mathcal{D}_{p,s}$, $\mathcal{D}_{p,f}$, $\mathcal{D}_{c,s}$, and $\mathcal{D}_{c,f}$.

Method	Visual Tool	Reflection	Update LLMs	Update Tools
ART [16]	✗	✗	Prompt	-
LATM [1]	✗	Global	Prompt	-
TRICE [18]	✗	Global	Instruction + RL	-
ToolkenGPT [7]	✗	-	✗	-
Toolformer [21]	✗	-	Fine-tune	-
VISPROG [6]	✓	✗	✗	✗
Visual ChatGPT [24]	✓	✗	✗	✗
InternGPT [13]	✓	✗	✗	✗
HuggingGPT [22]	✓	✗	✗	✗
ViperGPT [23]	✓	✗	✗	✗
ToT [8]	✓	✗	✗	✗
Chameleon [15]	✓	✗	✗	✗
ControlLLM [14]	✓	✗	✗	✗
MM-REACT [26]	✓	✗	✗	✗
Llava-plus [12]	✓	✗	Instruction	✗
Gorilla [17]	✓	✗	Instruction	✗
GPT4TOOLs [25]	✓	✗	Instruction	✗
OpenAGI [4]	✓	✗	Reinforcement Learning	✗
AssistGPT [3]	✓	Global	Prompt	✗
CLOVA (Ours)	✓	Global+Local	Prompt	Prompt

Table 1. Comparisons with representative tool-usage methods.

We use questions and inferred answers of incorrect cases (detailed in Section 3.4) to update the VQA tool. We also store correct cases with zero vectors as prompts. We use the language modeling loss [11] to train learnable prompts, where the Adam optimizer is used and the learning rate is $1e-3$. We train the prompts 100 steps for each instance.

4.1.3 Inference

The prompt ensemble process of the VQA tool has two steps. (1) We roughly select out 20 prompts from the prompt pool as candidates, by computing the similarity between the given query instance and stored instances in the prompt pool. (2) We use prompt ensemble (detailed in Section 3.4) to aggregate the 20 prompts for a query instance. In other words, we do not aggregate all stored prompts for a query instance, but 20 similar instances.

4.2. Update LOC tool

4.2.1 Model

In CLOVA, we use the OWL-ViT model [10] for object localization as the LOC tool. The architecture of OWL-ViT is shown in Fig. 11. OWL-ViT model uses a standard vision transformer as the image encoder and a similar transformer architecture as the text encoder. It removes the token pooling and final projection layer, and instead linearly projects each output token representation to obtain per-object image embeddings for classification. Besides, box coordinates are obtained by passing token representations through a small MLP. The text embeddings, which are called queries, are obtained by passing category names or other textual object descriptions through the text encoder. At inference time, given a set of candidate class names and an image, the model predicts a bounding box and a probability with which each query, and filters out the bounding box with the prediction confidence less than 0.1.

4.2.2 Training

In this study, upon identifying the need to update the LOC tool for learning a specific concept, we employ instance-wise prompts to update the OWL-ViT model. To achieve this, CLOVA first collect training data from open-vocabulary datasets. We adopt LVIS dataset [5] for OWL-ViT model. Taking the concept "glove" as an example, CLOVA randomly selects 200 samples from the LVIS dataset, whose class labels contain a glove as the training data. During training, visual features are extracted from the backbone, and CLOVA concatenates learnable prompts with the inputs to the vision transformer decoder. The model is trained with original losses introduced by OWL-ViT where only the prompts are learned. The losses include classification loss and bounding box regression loss. The former uses focal sigmoid cross-entropy [27] while the latter uses L_1 loss. For classification loss, we regard the

(a) Correct examples for plan generation

Instruction: An image shows one bare hand with the thumb on the right holding up a belly-first, head-up crab, with water in the background.
Plan:
Step1, Asking the left image, 'Does the image shows one bare hand with the thumb on the right holding a crab?'.
Step2, Asking the right image, 'Does the image shows one bare hand with the thumb on the right holding a crab?'.
Step3, Asking the left image, 'Is the crab belly-first and head-ups?'.
Step4, Asking the right image, 'Is the crab belly-first and head-ups?'.
Step5, Asking the left image, 'Is there water in the background?'.
Step6, Asking the right image, 'Is there water in the background?'.
Step7, Obtain whether the answer is 'yes' or 'no' by executing Python expression, based on the answers obtained in Step1, Step3, and Step5. If all answers obtained in Step1, Step3, and Step5 are 'yes', the answer in this step is 'yes'; On the contrary, the answer is 'no'.
Step8, Obtain whether the answer is 'yes' or 'no' by executing Python expression, based on the answers obtained in Step2, Step4, and Step6. If all answers obtained in Step2, Step4, and Step6 are 'yes', the answer in this step is 'yes'; On the contrary, the answer is 'no'.
Step9, Obtain whether the answer is 'yes' or 'no' by executing Python expression, based on the answers obtained in Step7 and Step8. If one of the two answers is equal to 'yes' and the rest one answer is equal to 'no', the answer in this step is 'yes'; On the contrary, the answer is 'no'.
Step10, Visualize results.

(b) Incorrect examples for plan generation

Instruction: There are two pairs of hands wearing gloves.
Plan:
Step1, Asking the left image, 'How many pairs of hands are in the image?'.
Step2, Asking the right image, 'How many pairs of hands are in the image?'.
Step3, Asking the left image, 'Are the hands wearing gloves?'.
Step4, Asking the right image, 'Are the hands wearing gloves?'.
Step5, Obtain whether the answer is 'yes' or 'no' by executing Python expression, based on the number obtained in Step1 and answers obtained in Step3. If the number is equal to '2' and the answer is equal to 'yes', the answer in this step is 'yes'; On the contrary, the answer is 'no'.
Step6, Obtain whether the answer is 'yes' or 'no' by executing Python expression, based on the number obtained in Step2 and answers obtained in Step4. If the number is equal to '2' and the answer is equal to 'yes', the answer in this step is 'yes'; On the contrary, the answer is 'no'.
Step7, Obtain whether the answer is 'yes' or 'no' by executing Python expression, based on the answers obtained in Step5 and Step6. If one of the two answers is equal to 'yes' and the rest one answer is equal to 'no', the answer in this step is 'yes'; On the contrary, the answer is 'no'.
Step8, Visualize results.
Reason: In Step5 of the plan, the plan judge whether there are two pairs of hands in the left image. It is wrong, the plan should judge whether the sum of pairs of hands in the two images are equal to two, and then judge whether the two pairs of hands are wearing gloves.

(c) Correct examples for program generation

Instruction: An image shows one bare hand with the thumb on the right holding up a belly-first, head-up crab, with water in the background.
Plan:
Step1, Asking the left image, 'Does the image shows one bare hand with the thumb on the right holding a crab?'.
Step2, Asking the right image, 'Does the image shows one bare hand with the thumb on the right holding a crab?'.
Step3, Asking the left image, 'Is the crab belly-first and head-ups?'.
Step4, Asking the right image, 'Is the crab belly-first and head-ups?'.
Step5, Asking the left image, 'Is there water in the background?'.
Step6, Asking the right image, 'Is there water in the background?'.
Step7, Obtain whether the answer is 'yes' or 'no' by executing Python expression, based on the answers obtained in Step1, Step3, and Step5. If all answers obtained in Step1, Step3, and Step5 are 'yes', the answer in this step is 'yes'; On the contrary, the answer is 'no'.
Step8, Obtain whether the answer is 'yes' or 'no' by executing Python expression, based on the answers obtained in Step2, Step4, and Step6. If all answers obtained in Step2, Step4, and Step6 are 'yes', the answer in this step is 'yes'; On the contrary, the answer is 'no'.
Step9, Obtain whether the answer is 'yes' or 'no' by executing Python expression, based on the answers obtained in Step7 and Step8. If one of the two answers is equal to 'yes' and the rest one answer is equal to 'no', the answer in this step is 'yes'; On the contrary, the answer is 'no'.
Step10, Visualize results.
Program:
ANSWER0=VQA(image=LEFT,question='Does the image shows one bare hand with the thumb on the right holding a crab?')
ANSWER1=VQA(image=RIGHT,question='Does the image shows one bare hand with the thumb on the right holding a crab?')
ANSWER2=VQA(image=LEFT,question='Is the crab belly-first and head-ups?')
ANSWER3=VQA(image=RIGHT,question='Is the crab belly-first and head-ups?')
ANSWER4=VQA(image=LEFT,question='Is there water in the background?')
ANSWER5=VQA(image=RIGHT,question='Is there water in the background?')
ANSWER6=EVAL(expr='{ANSWER0} and {ANSWER2} and {ANSWER4}')
ANSWER7=EVAL(expr='{ANSWER1} and {ANSWER3} and {ANSWER5}')
ANSWER8=EVAL(expr='{ANSWER6} xor {ANSWER7}')
FINAL_ANSWER=RESULT(var=ANSWER8)

(d) Incorrect examples for program generation

Instruction: A mitten is being worn in one image and the mittens are not being worn in the other image.
Plan:
Step1, Asking the left image, 'Is a mitten being worn in the image?'.
Step2, Asking the right image, 'Is a mitten being worn in the image?'.
Step3, Obtain whether the answer is 'yes' or 'no' by executing Python expression, based on the answers obtained in Step1 and Step2. If one of the two answers is equal to 'yes' and the rest one answer is equal to 'no', the answer in this step is 'yes'; On the contrary, the answer is 'no'.
Step4, Visualize results.
Program:
ANSWER0=VQA(image=LEFT,question='Is a mitten being worn in the image?')
ANSWER1=VQA(image=RIGHT,question='Is a mitten being worn in the image?')
ANSWER2=EVAL(expr='{ANSWER0} and {ANSWER1}')
FINAL_ANSWER=RESULT(var=ANSWER2)
Reason: The plan is correct, and can address the given question. But the Step3 of the program does not match the plan. The program should use 'xor' instead of 'and'.

Figure 2. Demonstration examples for multi-image reasoning tasks in $\mathcal{D}_{p,s}$, $\mathcal{D}_{p,f}$, $\mathcal{D}_{c,s}$, and $\mathcal{D}_{c,f}$.

(a) Correct examples for plan generation

Instruction: Hide the face of Nicole Kidman with face_with_tongue.
Plan:
Step1, Detect face regions from the given image, and obtain bounding boxes of face regions.
Step2, Select the face region of Nicole Kidman, based on the bounding boxes obtained in Step1.
Step3, Add the emoji face_with_tongue to the face region of Nicole Kidman in the given image, where the face region of Nicole Kidman is obtained in Step2.
Step4, Visualize results.

(b) Incorrect examples for plan generation

Instruction: Hide the face of Nicole Kidman with face_with_tongue.
Plan:
Step1, Detect face regions from the given image, and obtain bounding boxes of face regions.
Step2, Add the emoji face_with_tongue to the face regions the given image, where the face region is obtained in Step1.
Step4, Visualize results.
Reason: The plan is incorrect. In Step2, the program should use the SELECT model to classify face regions of Nicole Kidman, instead of adding emoji to all faces.

(c) Correct examples for program generation

Instruction: Hide the face of Nicole Kidman with face_with_tongue.
Plan:
Step1, Detect face regions from the given image, and obtain bounding boxes of face regions.
Step2, Select the face region of Nicole Kidman, based on the bounding boxes obtained in Step1.
Step3, Add the emoji face_with_tongue to the face region of Nicole Kidman in the given image, where the face region of Nicole Kidman is obtained in Step2.
Step4, Visualize results.
Program:
OBJ0=FACEDET(image=IMAGE)
OBJ1=SELECT(image=IMAGE,object=OBJ0,query='Nicole Kidman',category=None)
IMAGE0=EMOJI(image=IMAGE,object=OBJ1,emoji='face_with_tongue')
FINAL_RESULT=RESULT(var=IMAGE0)

(d) Incorrect examples for program generation

Instruction: Hide the face of Nicole Kidman with face_with_tongue.
Plan:
Step1, Detect face regions from the given image, and obtain bounding boxes of face regions.
Step2, Select the face region of Nicole Kidman, based on the bounding boxes obtained in Step1.
Step3, Add the emoji face_with_tongue to the face region of Nicole Kidman in the given image, where the face region of Nicole Kidman is obtained in Step2.
Step4, Visualize results.
Program:
OBJ0=FACEDET(image=IMAGE)
OBJ1=SELECT(image=IMAGE,object=OBJ0,query='Nicole Kidman',category=None)
IMAGE0=REPLACE(image=IMAGE,object=OBJ1,emoji='face_with_tongue')
FINAL_RESULT=RESULT(var=IMAGE0)
Reason: The plan is correct, and can address the given question. But the Step3 of the program does not match the plan. The program should use EMOJI functions instead of REPLACE.

Figure 3. Demonstration examples for image editing tasks in $\mathcal{D}_{p,s}$, $\mathcal{D}_{p,f}$, $\mathcal{D}_{c,s}$, and $\mathcal{D}_{c,f}$.

learned concept and one randomly selected class name in the image as positive labels and randomly select 13 class names as negative labels per image to avoid overfitting. The number of learned prompts is 100, and the prompt is randomly initialized. We set the maximum training step as 100. If within 100 steps, the sample could be detected correctly by the model, CLOVA would save this sample and the prompt, otherwise, CLOVA will remove this sample and the corresponding prompt. Only if all the positive labels are correctly classified and the average IOU between the prediction boxes and ground-truth boxes is larger than 0.6, we recognize this sample is correctly detected. We use Adam as the optimizer and set the learning rate as $5e-4$. Eventually, we save the feature and the learned prompt of each correctly detected instance for this concept.

4.2.3 Inference

The prompt ensemble process of the LOC tool also has two steps. (1) Given a visual concept and a query image, we select all the prompts having the same visual concept from the prompt pool as candidates. We then filter out the prompts by making predictions with each candidate prompt, if the prediction confidence is larger than 0.1, we will use the prompt for the query image, otherwise, we will remove this candidate prompt. (2) We use prompt ensemble (detailed in Section 3.4) to aggregate all the selected prompts for a query

(a) Correct examples for plan generation

Instruction: Tag the wild animals that lives on the land
Plan:
Step1, Locate wild animals from the given image, and obtain bounding boxes of wild animals.
Step2, List 'wild animals that lives on the land' by asking GPT.
Step3: Classify wild animals that lives on the land, based on the bounding boxes of wild animals obtained in Step1 and the wild animal list obtain in Step2.
Step4: Tag the bounding boxes and labels to the classified wild animals.
Step5: Visualize results.

(b) Incorrect examples for plan generation

Instruction: Tag two utensils used for drinking, which are usually in the kitchen
Plan:
Step1, Locate utensil from the given image, and obtain bounding boxes of utensil.
Step2, List two 'utensils used for drinking' by asking GPT.
Step3: Classify utensil, based on the bounding boxes of utensil obtained in Step1 and the utensil list obtain in Step2.
Step4: Tag the bounding boxes and labels to the classified utensil.
Step5: Visualize results.
Reason: In Step2, it should list 'utensils used for drinking, which are usually in the kitchen', instead of only 'utensils used for drinking'.

(c) Correct examples for program generation

Instruction: Tag the wild animals that lives on the land
Plan:
Step1, Locate wild animals from the given image, and obtain bounding boxes of wild animals.
Step2, List 'wild animals that lives on the land' by asking GPT.
Step3: Classify wild animals that lives on the land, based on the bounding boxes of wild animal obtained in Step1 and the wild animals list obtain in Step2.
Step4: Tag the bounding boxes and labels to the classified wild animals.
Step5: Visualize results.
Program:
OBJ0=LOC(image=IMAGE,object='wild animal')
LIST0=LIST(query='wild animals that lives on the land',max=20)
OBJ1=CLASSIFY(image=IMAGE,object=OBJ0,categories=LIST0)
IMAGE0=TAG(image=IMAGE,object=OBJ1)
FINAL_RESULT=RESULT(var=IMAGE0)

(d) Incorrect examples for program generation

Instruction: Tag the wild animals that lives on the land
Plan:
Step1, Locate wild animals from the given image, and obtain bounding boxes of wild animals.
Step2, List 'wild animals that lives on the land' by asking GPT.
Step3: Classify wild animals that lives on the land, based on the bounding boxes of wild animal obtained in Step1 and the wild animals list obtain in Step2.
Step4: Tag the bounding boxes and labels to the classified wild animals.
Step5: Visualize results.
Program:
OBJ0=LOC(image=IMAGE,object='wild animal')
LIST0=LIST(query='wild animals that lives on the land',max=20)
OBJ1=LOC(image=IMAGE,object=land')
IMAGE0=TAG(image=IMAGE,object=OBJ1)
FINAL_RESULT=RESULT(var=IMAGE0)
Reason: The plan is correct, but the program does not match the plan. In Step3 in the program, it is wrong to use the LOC model. It should use the CLASSIFY for OBJ0 obtained in Step1.

Figure 4. Demonstration examples for factual knowledge taggings task in $\mathcal{D}_{p,s}$, $\mathcal{D}_{p,f}$, $\mathcal{D}_{c,s}$, and $\mathcal{D}_{c,f}$.

instance and contact the prompt with the input to produce a prediction.

4.3. Update SEG tool

4.3.1 Model

We use the Maskformer [2] model for the SEG tool. One Maskformer model is composed of three components: a backbone, a pixel decoder, and a transformer decoder. The backbone extracts features of images. Then, the pixel decoder gradually upsamples image features to extract per-pixel embeddings. Finally, the transformer decoder uses image features with our learnable prompts to generate per-mask embeddings that are combined with pre-pixel embedding for mask prediction. In our prompt tuning scheme, the visual features are extracted from the backbone followed by average pooling, whose dimension is 256. We concatenate learnable prompts with the inputs to the transformer

You are a **planner**. Given a question, you need to generate the plan.

Some correct cases are as follows.
Instruction : What color is the curtain that is to the right of the mirror?
Plan:
Step1, Locate mirror in the given image, and obtain bounding boxes of mirror.
Step2, Crop the region on the right side of the mirror from the given image, based on the bounding boxes of mirror. The bounding boxes are obtained in Step1.
Step3, Asking the image region on the right side of the mirror, 'What color is the curtain?'. The image region is cropped in Step2.
Step4, Visualize results.

Some incorrect cases and their reasons are as follows.
Instruction : Who is in the blue water?
Plan:
Step1, Locate blue regions in the given image, and obtain bounding boxes of the blue regions.
Step2, Crop the blue regions from the given image, based on the bounding boxes of the blue regions. The bounding boxes are obtained in Step1.
Step3, Locate people in the blue regions of the given image, and obtain bounding boxes of people. The blue regions are cropped in Step2.
Step4, Count the number of people, based on bounding boxes of people. The bounding boxes are obtained in Step3.
Step5, Obtain the names of the people in the blue water, based on the number of people. The number is obtained in Step4. If there is only one person, provide their name; if there are multiple people, provide a list of their names.
Step6, Visualize results.
Reason:
The Step4-Step6 in the plan is correct. After obtaining the bounding boxes of people in Step3, the plan should crop the image region of the person, and ask who is the person, instead of counting the number.

Now, you need to generate plan for the following query.
Instruction : Is there any snow or grass in this scene?
Plan:

Figure 5. Prompts for plan generation.

You are a **programmer**. You need to generate the program based on Instruction and Plan.

Some **correct cases** are as follows.
Instruction: What color is the curtain that is to the right of the mirror?
Plan:
Step1, Locate mirror in the given image, and obtain bounding boxes of mirror.
Step2, Crop the region on the right side of the mirror from the given image, based on the bounding boxes of mirror. The bounding boxes are obtained in Step1.
Step3, Asking the image region on the right side of the mirror, 'What color is the curtain?'. The image region is cropped in Step2.
Step4, Visualize results.
Program:
BOX0=LOC(image=IMAGE,object='mirror')
IMAGE0=CROP_RIGHTOF(image=IMAGE,box=BOX0)
ANSWER0=VQA(image=IMAGE0,question='What color is the curtain?')
FINAL_RESULT=RESULT(var=ANSWER0)...

Some **incorrect cases** and their reasons are as follows.
Instruction : What is the red piece of clothing in this photograph?
Plan:
Step 1: Locate the red piece of clothing in the given photograph and obtain the bounding box of the clothing.
Step 2: Crop the region of the red piece of clothing from the given photograph based on the bounding box obtained in step 1.
Step 3: Asking the image region of the red piece of clothing, "What is the red piece of clothing?"
Step 4: Visualize results.
Program:
BOX0=LOC(image= IMAGE,object='red piece of clothing')
IMAGE0=CROP(image= IMAGE,box=BOX0)
ANSWER0=VQA(image=IMAGE1,question='What is the red piece of clothing?')
FINAL_RESULT=RESULT(var=ANSWER0)
Reason: The Step3 in the program have bug, because the variable IMAGE1 is not defined. It should be IMAGE0.

Now, you need to generate program for the following query.
Instruction: Is the baseball man to the right or to the left of the woman?
Plan:
Step1, Locate the baseball man in the given image, and obtain bounding boxes of the baseball man.
Step2, Locate the woman in the given image, and obtain bounding boxes of the woman.
Step3, Obtain the relative position of the baseball man and the woman, based on the bounding boxes of the baseball man and the woman. If the baseball man is to the right of the woman, the answer is 'right'; On the contrary, the answer is 'left'.
Step4, Visualize results.
Program:

Figure 6. Prompts for program generation

You are a **debugger**. You need to check which model cause of the wrong answer. Errors may exist in the plan, program, or functions called by the program.

Instruction: Is the lamp different in color than the shirt?
Description of the Input Image: a photography of a couple of people in a restaurant
Human Feedback: The correct answer is yes
Our Wrong Answer: no
Following are the decomposed plan, used program, and obtained result in each step.
Plan:
Step1, Locate the lamp in the given image, and obtain bounding boxes of lamp.
Step2, Crop the region of the lamp from the given image, based on bounding boxes of lamp. The bounding boxes are obtained in Step1.
Step3, Asking the image region of lamp, 'What color is the lamp?'. The image region of lamp is cropped in Step2.
Step4, Locate the shirt in the given image, and obtain bounding boxes of shirt.
Step5, Crop the region of the shirt from the given image, based on bounding boxes of shirt. The bounding boxes are obtained in Step4.
Step6, Asking the image region of shirt, 'What color is the shirt?'. The image region of lamp is cropped in Step5.
Step7, Obtain whether the answer is 'yes' or 'no', based on the color of lamp and the color of shirt. The color of lamp and shirt is obtained in Step3 and Step6, respectively. If their color are the same, the answer is 'yes'; On the contrary, the answer is 'no'.
Step8, Visualize results.
Program and obtained result in each step:
Step1 Program: BOX0=LOC(image=IMAGE,object='lamp')
Result of The coordinate of BOX0: [[45, 78, 245, 345]]
Step2 Program: IMAGE0=CROP(image=IMAGE,box=BOX0)
Result of The description of IMAGE0: a photography of a couple of people on a snowboard in the snow
Step3 Program: BOX1=LOC(image=IMAGE0,object='shirt')
Result of BOX1 is empty
Step4 Program: ANSWER0=COUNT(box=BOX1)
Result of ANSWER0: 0
Step5 Program: ANSWER1=EVAL(expr="'yes' if {ANSWER0} > 0 else 'no'")
Result of ANSWER1: no
Step6 Program: FINAL_RESULT=RESULT(var=ANSWER1)
Result of FINAL_RESULT: no
Error Location: program
Reason: The plan are correct, and can address the given question. But the Step3-Step6 of the program does not match the plan. The plan locates, crops, and asks color of the lamp and shirt, but the program counts the number of lamp.

The failed case needs to be debugged is as follows.
Instruction: Is the wall behind a boy?
Description of the Input Image: a photography of a man holding a wii remote in his hand
Human Feedback: The correct answer is no
Our Wrong Answer: yes
Following are the decomposed plan, used program, and obtained result in each step.
Plan:
Step1, Locate the boy in the given image, and obtain bounding boxes of the boy.
Step2, Crop the image region behind the boy from the given image, based on bounding boxes of the boy. The bounding boxes are obtained in Step1.
Step3, Locate the wall in the region behind the boy, and obtain bounding boxes of the wall. The region behind the boy is cropped in Step2.
Step4, Count the number of walls, based on bounding boxes of walls. The bounding boxes are obtained in Step3.
Step5, Obtain whether the answer is 'yes' or 'no', based on the number of walls. The number is obtained in Step4. If the number is greater than zero, the answer is 'yes'; On the contrary, the answer is 'no'.
Step6, Visualize results.
Program and obtained result in each step:
Step1 Program: BOX0=LOC(image=IMAGE,object='boy')
The coordinate of BOX0: [[100, 43, 414, 374]]
Step2 Program: IMAGE0=CROP_BEHIND(image=IMAGE,box=BOX0)
The description of IMAGE0: a photography of a man holding a wii remote in his hand
Step3 Program: BOX1=LOC(image=IMAGE0,object='wall')
The coordinate of BOX1: [[279, 1, 498, 207], [30, 1, 498, 207]]
Step4 Program: ANSWER0=COUNT(box=BOX1)
Result of ANSWER0: 2
Step5 Program: ANSWER1=EVAL(expr="'yes' if {ANSWER0} > 0 else 'no'")
Result of ANSWER1: yes
Step6 Program: FINAL_RESULT=RESULT(var=ANSWER1)
Result of FINAL_RESULT: yes
Error Location:
Reason:

Figure 7. Prompts for global reflection.

decoder. The architecture of Maskformer is shown in Fig. 12.

4.3.2 Training

The size of learnable prompts is $\mathbb{R}^{100\times256}$, that is we learn 100 vectors as prompts, and the dimension of each one is

You are a **debugger**. You need to check which model cause of the wrong answer. After given one step, you need to determine if this step is correct. If it is incorrect, you need to provide the error location, and explain the reason.

The failed case is as follows.
Instruction: On which side of the photo is the stuffed bear?
Description of the Input Image: a photography of a display case with teddy bears
Desirable Answer: right
Our Wrong Answer: left
It has totally 4 steps. Step1 have been checked:
Step1
Plan: Locate stuffed bear in the given image
Program: BOX0=LOC(image=IMAGE, object='stuffed bear')
Result of The coordinate of BOX0: [[136, 58, 184, 116], [191, 87, 227, 135]]]

Now you need to check Step2.
Plan: Crop the region of stuffed bear from the given image.
Program: IMAGE0=CROP(image=IMAGE,box=BOX0)
The description of IMAGE0: a photography of a group of stuffed animals sitting next to each other.

Is this Step2 correct? Errors may exist in the plan, program, or functions called by the program If they are correct, directly output "yes" and do not output any other content. If incorrect, firstly output "no", then provide the error location and reason.

Figure 8. Prompts for local reflection.

256. We remove the classification loss of Maskformer, and only use the mask loss [2] to train learnable prompts, where the Adam optimizer is used and the learning rate is $1e-1$. Given a visual concept that needs to be learned, we randomly select 50 samples having this visual concept from the LVIS dataset [5]. We train the prompts 100 steps for each instance. After training, we store the concept name, and image features of instances, and learned prompts of image features in the the prompt pool.

4.3.3 Inference

The prompt ensemble process of the SEG tool also has two steps. (1) Given a visual concept, we roughly select out 10 prompts having the same visual concept from the prompt pool as candidates, by computing the similarity between the given query instance and stored instances in the prompt pool. (2) We use prompt ensemble (detailed in Section 3.4) to aggregate the 10 prompts for a query instance. In other words, we do not aggregate all stored prompts for a query instance, but 10 similar instances.

4.4. Update SELECT and CLASSIFY tools

4.4.1 Model

We utilize CLIP [19] as the SELECT and CLASSIFY tools. The model consists of a Vision Transformer (ViT) as the image encoder and a Transformer-based text encoder. The image encoder and text encoder encode images and text descriptions into high-dimensional feature vectors respectively. It learns to align the representations of related content and separate unrelated content in the embedding space by utilizing a contrastive loss function. This loss function encourages CLIP to maximize the similarity between corresponding image-text pairs while minimizing it for non-corresponding pairs. In the prompt tuning scheme, learnable tokens are introduced into the image encoder and fine-tuned. These prompts are stored in the prompt pool for later use. During inference, the prompt is selected from the prompt pool and replaced with the image encoder to improve the precision of query data. The architecture of CLIP is shown in Fig. 13.

4.4.2 Training

We update the CLIP through the deep prompt tuning [9]. We utilize the Adam optimizer with a learning rate set to $1e-2$. To obtain training data, CLOVA uses the concept to be learned to automatically retrieve 7 images from Google Images as positive data through web scraping. The first image is used for validation, while the remaining ones are used for training. Furthermore, CLOVA derives additional concepts related to but different from the target concept through GPT and subsequently collected data associated with these concepts as negative samples. CLOVA then uses both the positive data and negative data for training. During the training phase, we introduce 100 learnable prompts for each of the first three layers of the vision transformer to facilitate prompt tuning. We conduct prompt tuning for 100 steps per instance, followed by a validation step. If the accurate prediction is achieved on the validation data, CLOVA systematically stores features of crawled images and learned prompts in the prompt pool, otherwise, these learned prompts are discarded.

4.4.3 Inference

The prompt ensemble process of the SELECT and CLASSIFY tools also has two steps. (1) Given query data, we select data with its feature and prompt from the prompt pool

You are an **inference maker**. Your goal is that, given a failed case including a question, its desirable answer, our wrong answer, our plan, our programs, and the analysis about the wrong step, you need to infer the desirable intermediate results of the wrong step.

Following are some inferring examples.
Instruction: It there three bottles on the table?
The description of Input image: a photography of a restaurant with a table set
Human Feedback: The correct answer is yes
Our Wrong Answer: no
Following are the plan, used program, and obtained result in each step.
Plan:
Step1, Locate table in the given image, and obtain bounding boxes of table.
Step2, Crop the region of table from the given image, based on bounding boxes of table. The bounding boxes are obtained in Step1.
Step3, Asking the image region of table, 'How many bottles?'. The image region of the table is obtained in Step2.
Step4, Obtain the answer, based on the intermediate answers obtained in Step3.
Step5, Visualize results.
Program and obtained result in each step:
Step1
Program: BOX0=LOC(image=IMAGE,object='table')
The coordinate of BOX0: [[40, 240, 581, 479]
Step2
Program: IMAGE0=CROP(image=IMAGE,box=BOX0)
The description of IMAGE0: a photography of a table set with a white table cloth and red plates
Step3
Program: ANSWER0=VQA(image=IMAGE0,question= 'How many bottles?')
Results of ANSWER0: two
Step4
Program: ANSWER1=EVAL(expr="'yes' if {ANSWER0} == three' else 'no'")
Result of ANSWER1: no
Step5:
Program: FINAL_RESULT=RESULT(var=ANSWER1)
Result of FINAL_RESULT: no
Error Location: functions called by programs
Reason: In the Step3 of the program, the used function 'VQA' failed to count the number of bottles, as the obtained result of ANSWER0 is 'two' instead of 'three'.
Correct answer of the wrong step: three

Now, you will be given the failed case that needs to infer, as follows. Based on the expected answer, the intermediate results we got at each step of the program, and the analysis of the wrong step. You need to inference the desirable intermediate results of the wrong VQA step.
Question: What color is the dog, brown or red?
Description of the Input Image: a photography of a group of motorcycles parked in a field
Desirable Answer: red
Our Wrong Answer: brown
Following are the decomposed plan, used program, and obtained result in each step.
Plan:
Step1, Locate the dog in the given image and obtain bounding boxes of the dog.
Step2, Crop the region where the dog is located from the given image, based on the bounding boxes of the dog. The bounding boxes are obtained in Step1.
Step3, Asking the image region of dog, 'What color is the dog?'. The image region of the table is obtained in Step2.
Step4, Visualize results.
Program and obtained result in each step:
Step1
Program: BOX0=LOC(image=IMAGE,object='dog')
The coordinate of BOX0: [[51, 65, 83, 96]]
Step2
Program: IMAGE0=CROP(image=IMAGE,box=BOX0)
The description of IMAGE0: a photography of a dog is sniffing a toy in the grass
Step3
Program: ANSWER0=VQA(image=IMAGE0,question='What color is the dog?')
Result of ANSWER0: brown
Step4:
Program: FINAL_RESULT=RESULT(var=ANSWER0)
Result of FINAL_RESULT: brown
Error Location:
functions called by programs
Reason:
In Step3, the function 'VQA' failed to correctly infer the color of the dog based on the cropped region. The obtained result of ANSWER0 is 'brown', which is not one of the expected colors 'red'
Correct answer of the wrong step:

Figure 9. Prompts of inferring answers for the VQA tool.

with the same concept as the query data. Subsequently, we compute the similarity between the query data and the selected data. The prompts corresponding to data with high similarities are used for the query data. (2) We use prompt ensemble (detailed in Section 3.4) to aggregate selected prompts for the query data.

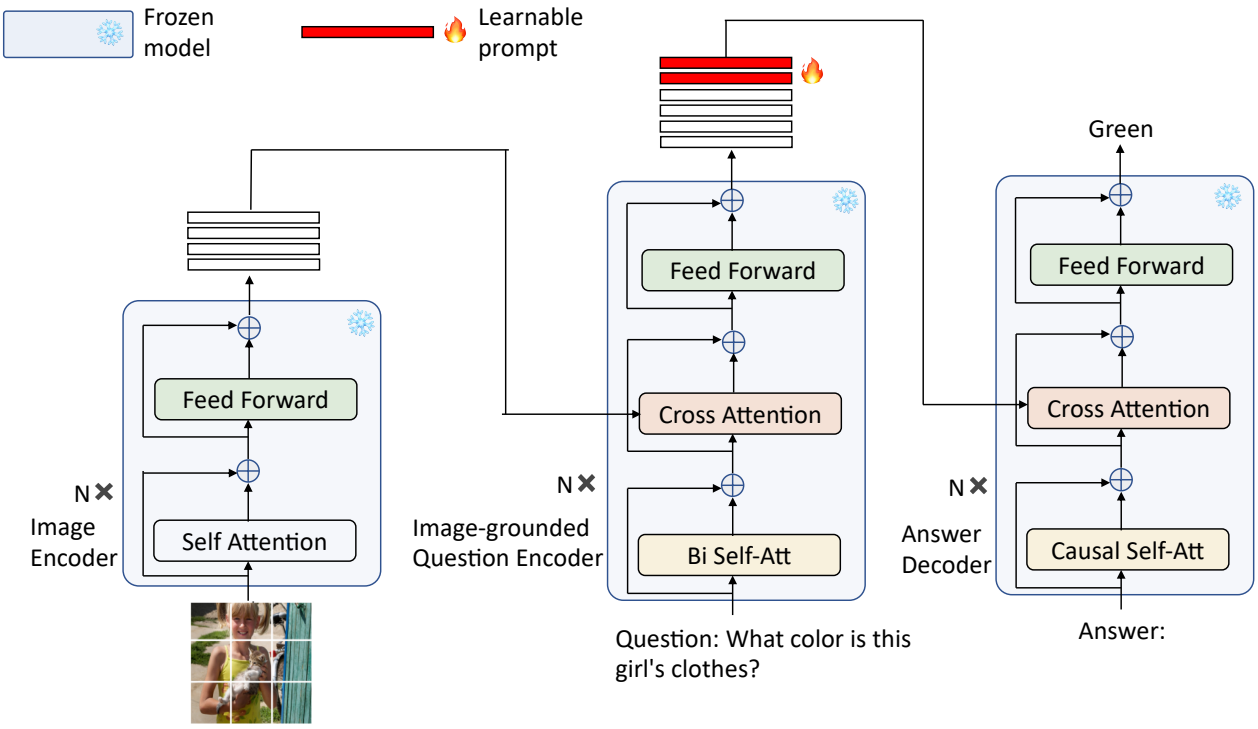

Figure 10. The architecture of BLIP.

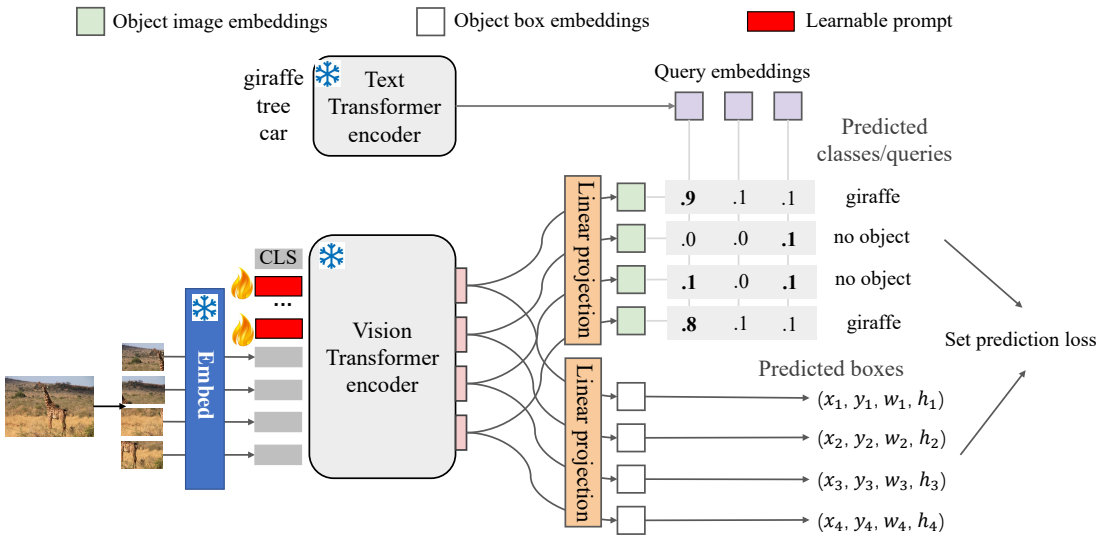

Figure 11. The architecture of OWL-ViT.

4.5. Update REPLACE tool

We use Stable Diffusion [20] as the REPLACE tool. In Stable Diffusion, the architecture comprises four key components: Sampler, Variational Autoencoder (VAE), UNet, and CLIPEmbedder. The Sampler and UNet focus on the actual image generation, the VAE provides a deep understanding of image content, and the CLIPEmbedder ensures the relevance and accuracy of the generated images in relation to the text inputs. We added learnable prompts to the text encoder of the CLIPEmbedder component. During the prompt tuning phase, we tune a prompt for training images and store it. The dimensionality of the prompt is 768. The architecture of Stable Diffusion is shown in Fig. 14.

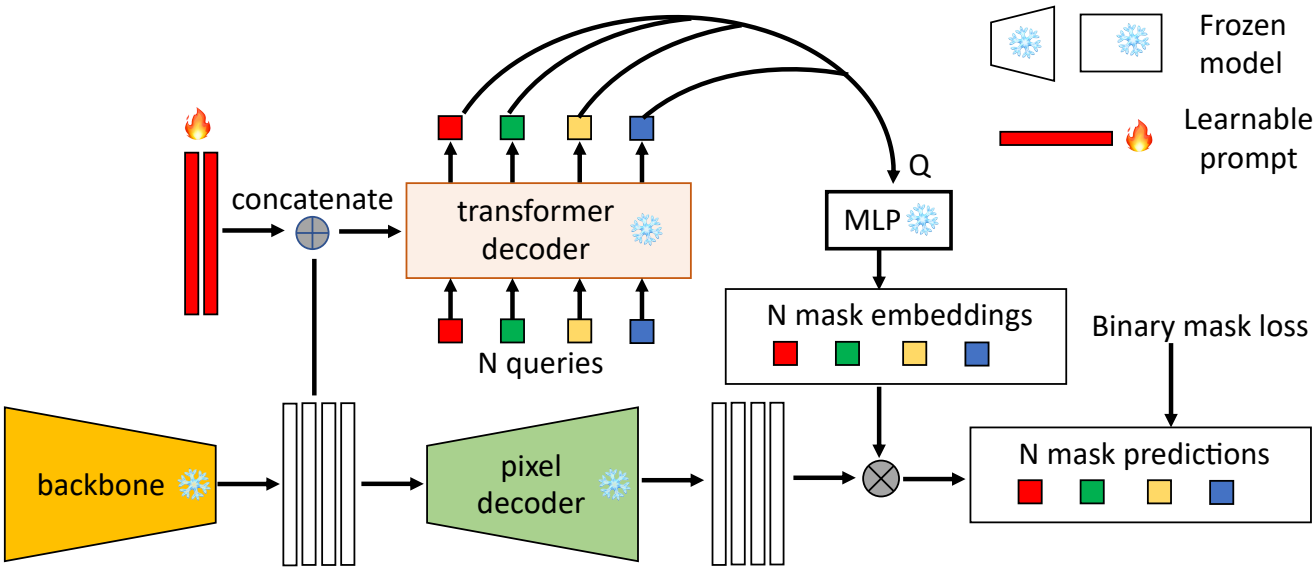

Figure 12. The architecture of Maskformer.

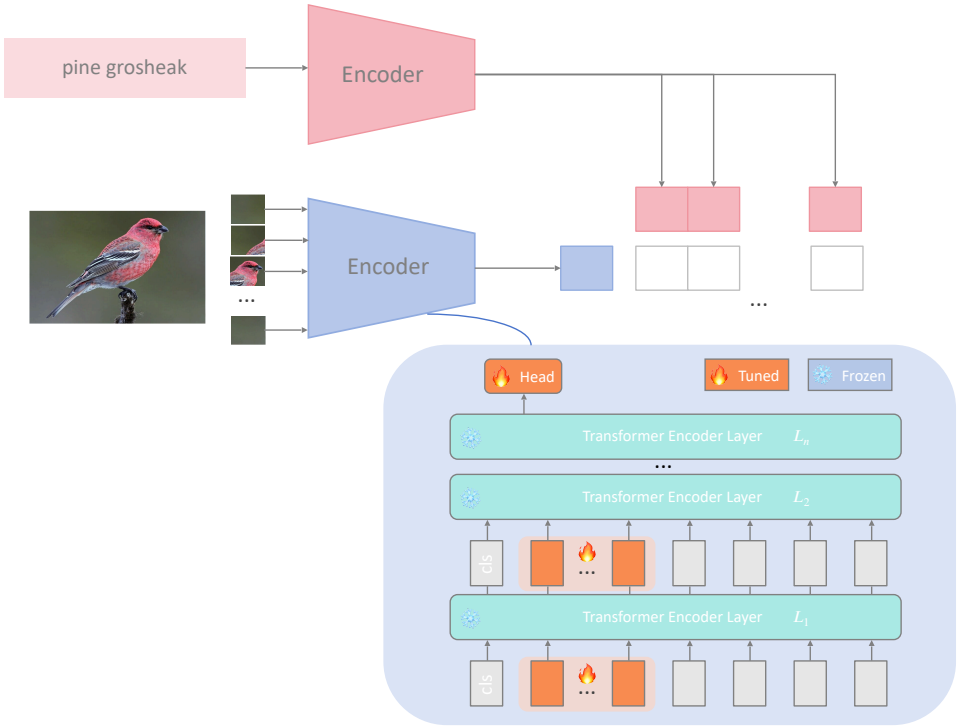

Figure 13. The architecture of CLIP.

4.5.1 Training

In order to facilitate the learning of a specific concept for the Stable Diffusion model, CLOVA employs web scraping techniques to retrieve 7 images representing this concept from Google Images. We train prompts in the text encoder of the Stable Diffusion model using downloaded images. During the training process, the Latent Diffusion Model(LDM) loss [20] is minimized. Subsequently, the 768-dimensional prompt obtained from the training stage is stored in the prompt pool. In terms of experimental setup,

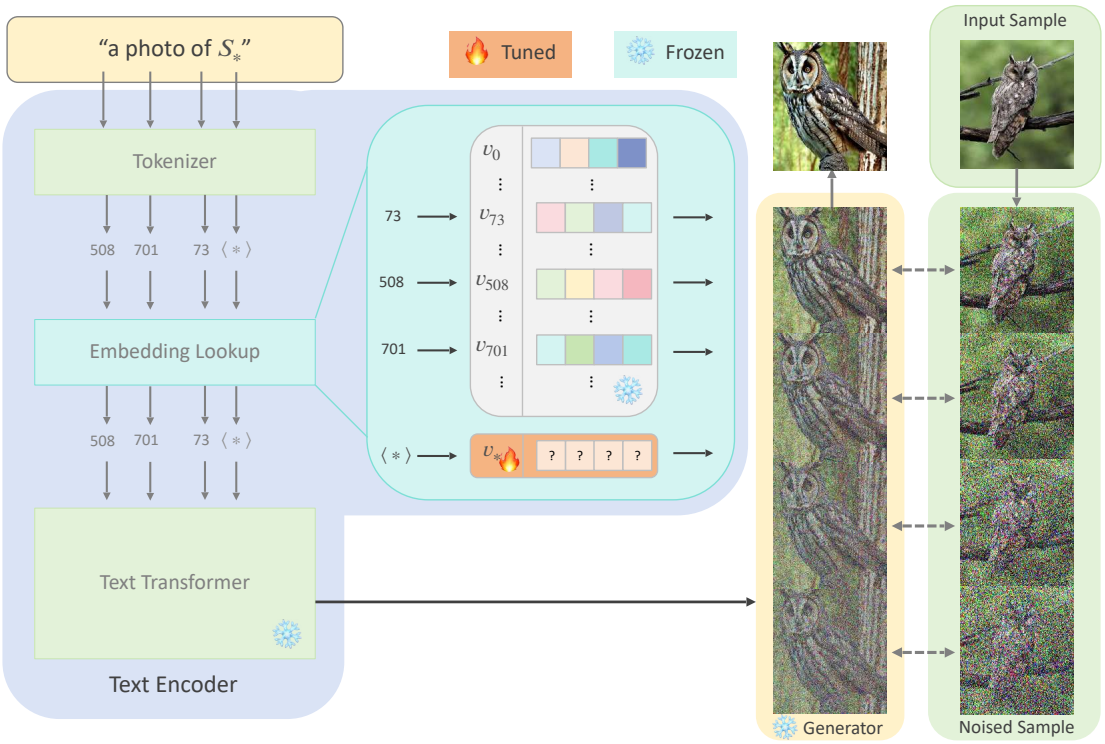

Figure 14. The architecture of Stable Diffusion.

we employ the Adam optimizer. Through our experimentation, we find that setting the learning rate to $5e-3$ achieves the best learning results.

4.5.2 Inference

Similar to other visual tools, inference for the Stable Diffusion model also contains two steps. Given query data, CLOVA first locates prompts corresponding to the concept in the prompt pool and then loads the prompt into the text encoder of the Stable Diffusion model. Based on the query and mask, we perform editing on the input image.

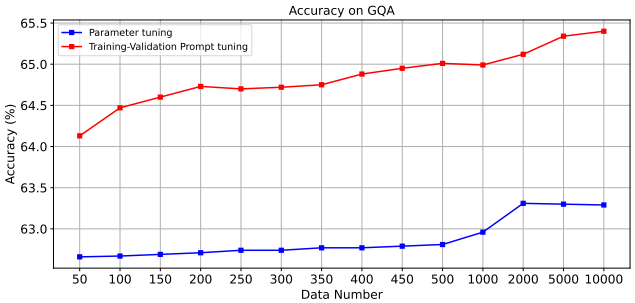

Figure 15. Accuracy curves on the GQA dataset

5. More Experimental Results

5.1. Training-validation prompt tuning for the VQA tool

We further evaluate the proposed validation-learning prompt tuning scheme for the VQA tool, where experiments are conducted on the compositional VQA task using the GQA dataset. We use the BLIP model for the GQA dataset and compare our prompt tuning scheme with direct tuning parameters. We report accuracies with using different numbers of training data. Results are shown in Fig. 15. Our method has higher performance throughout the entire training process, no matter whether the number of training data is small or large, showing the effectiveness of the proposed validation-learning prompt tuning scheme for the VQA tool.

5.2. Evaluation on the online setting

CLOVA can be applied to a more practical online learning setting. In this case, CLOVA is evaluated in a dynamic data stream. If it makes a correct prediction on a task, only the inference phase is activated, and the task is tagged as a correct prediction; if it makes an incorrect prediction on a task, this task is tagged as a wrong prediction, and the reflection and learning phases are activated to update tools. After the data stream, we calculate the accuracy based on tagged predictions of all cases. We conduct experiments on the compositional VQA and multi-image reasoning tasks, where the

Dataset	Method	LLama2-7B	GPT-3.5-turbo	GPT-4
GQA	Baseline	39.2	46.4	52.6
	+ Update LLMs	44.8	51.0	55.4
	+ Update visual tools	50.2	53.0	57.8
NLVRv2	Baseline	50.0	60.2	64.8
	+ Update LLMs	57.4	61.0	66.4
	+ Update visual tools	61.6	62.6	67.4

Table 2. Different LLMs on the online learning setting using the GQA and NLVRv2 datasets.

GQA and NLVRv2 datasets are used. Results are shown in Tab. 2. Similar to the offline setting in Section 4.2, updating LLMs and visual tools both leads to improvements.

5.3. More case studies

We provide more cases to show the reflection and learning phases of CLOVA. The reflection and learning phases for LLMs are shown in Fig. 16. The reflection and learning phases for the SELECT tool are shown in Fig. 17. The reflection and learning phases for the LOC tool are shown in Fig. 18. The reflection and learning phases for the REPLACE tool are shown in Fig. 19. The reflection and learning phases for the CLASSIFY tool are shown in Fig. 20. The reflection and learning phases for the SEG tool are shown in Fig. 21.

References

[1] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *ICLR*, 2024.

[2] Bowen Cheng, Alex Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. In *NeurIPS*, pages 17864–17875, 2021.

[3] Difei Gao, Lei Ji, Luowei Zhou, Kevin Qinghong Lin, Joya Chen, Zihan Fan, and Mike Zheng Shou. Assistgpt: A general multi-modal assistant that can plan, execute, inspect, and learn. *arXiv preprint arXiv:2306.08640*, 2023.

[4] Yingqiang Ge, Wenyue Hua, Jianchao Ji, Juntao Tan, Shuyuan Xu, and Yongfeng Zhang. Openagi: When llm meets domain experts. In *NeurIPS*, pages 5539–5568, 2023.

[5] Agrim Gupta, Piotr Dollar, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation. In *CVPR*, pages 5356–5364, 2019.

[6] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *CVPR*, pages 14953–14962, 2023.

[7] Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. In *NeurIPS*, pages 45870–45894, 2023.

[8] Pengbo Hu, Ji Qi, Xingyu Li, Hong Li, Xinqi Wang, Bing Quan, Ruiyu Wang, and Yi Zhou. Tree-of-mixed-thought: Combining fast and slow thinking for multi-hop visual reasoning. *arXiv preprint arXiv:2308.09658*, 2023.

[9] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *ECCV*, pages 709–727, 2022.

[10] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. In *NeurIPS*, pages 39648–39677, 2023.

[11] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *ICML*, pages 12888–12900, 2022.

[12] Shilong Liu, Hao Cheng, Haotian Liu, Hao Zhang, Feng Li, Tianhe Ren, Xueyan Zou, Jianwei Yang, Hang Su, Jun Zhu, Lei Zhang, Jianfeng Gao, and Chunyuan Li. Llava-plus: Learning to use tools for creating multimodal agents. *2311.05437,arXiv*, 2023.

[13] Zhaoyang Liu, Yinan He, Wenhai Wang, Weiyun Wang, Yi Wang, Shoufa Chen, Qinglong Zhang, Yang Yang, Qingyun Li, Jiashuo Yu, et al. Interngpt: Solving vision-centric tasks by interacting with chatbots beyond language. *arXiv preprint arXiv:2305.05662*, 2023.

[14] Zhaoyang Liu, Zeqiang Lai, Gao Zhangwei, Erfei Cui, Zhiheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and Wang Wenhai. Controlllm: Augment language models with tools by searching on graphs. *arXiv preprint arXiv:2305.10601*, 2023.

[15] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In *NeurIPS*, pages 43447–43478, 2023.

[16] Bhargavi Paranjape, Scott M. Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *ArXiv*, abs/2303.09014, 2023.

[17] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

[18] Shuofei Qiao, Honghao Gui, Huajun Chen, and Ningyu Zhang. Making language models better tool learners with execution feedback. *ArXiv*, abs/2305.13068, 2023.

[19] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, pages 8748–8763, 2021.

[20] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10684–10695, 2022.

[21] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, pages 68539–68551, 2023.

[22] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. In *NeurIPS*, pages 38154–38180, 2023.

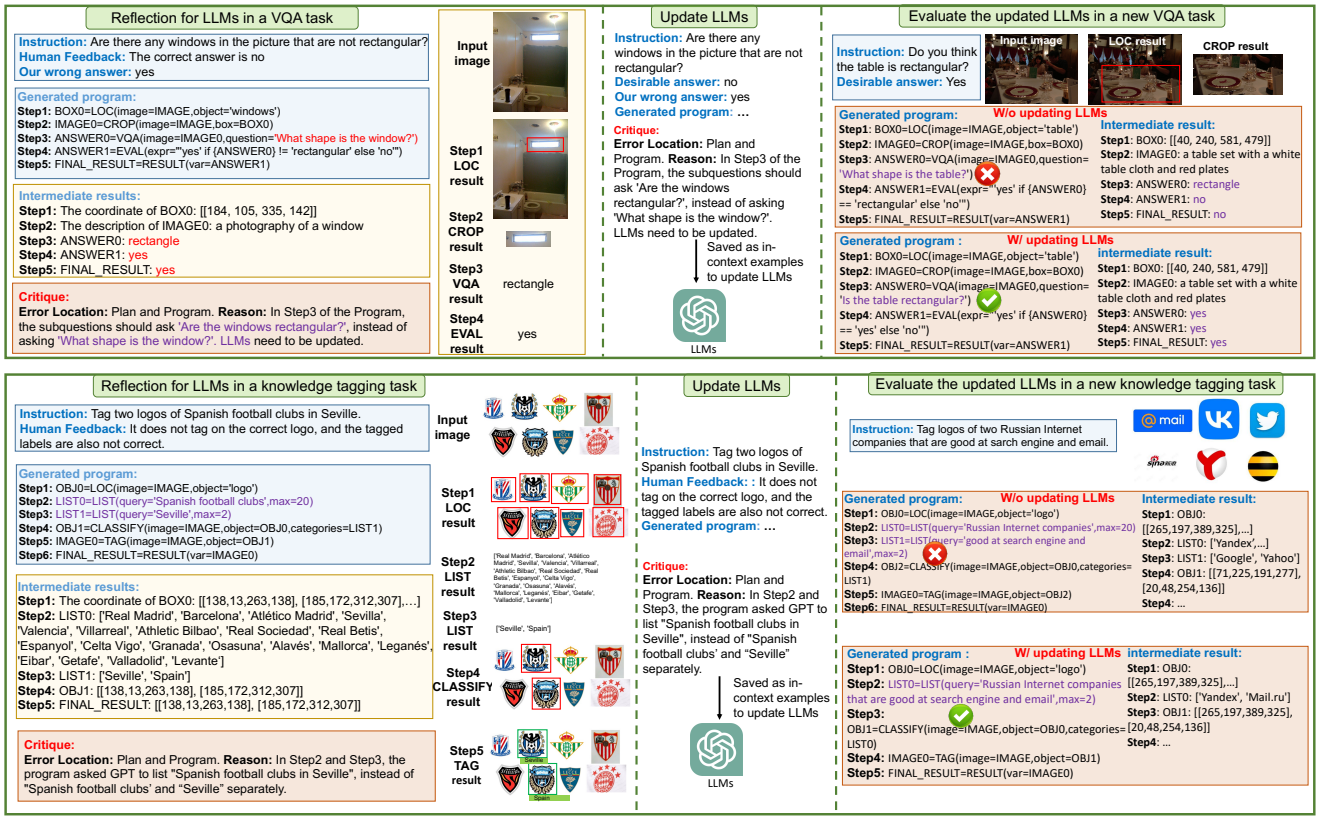

Figure 16. Case studies of updating LLMs.

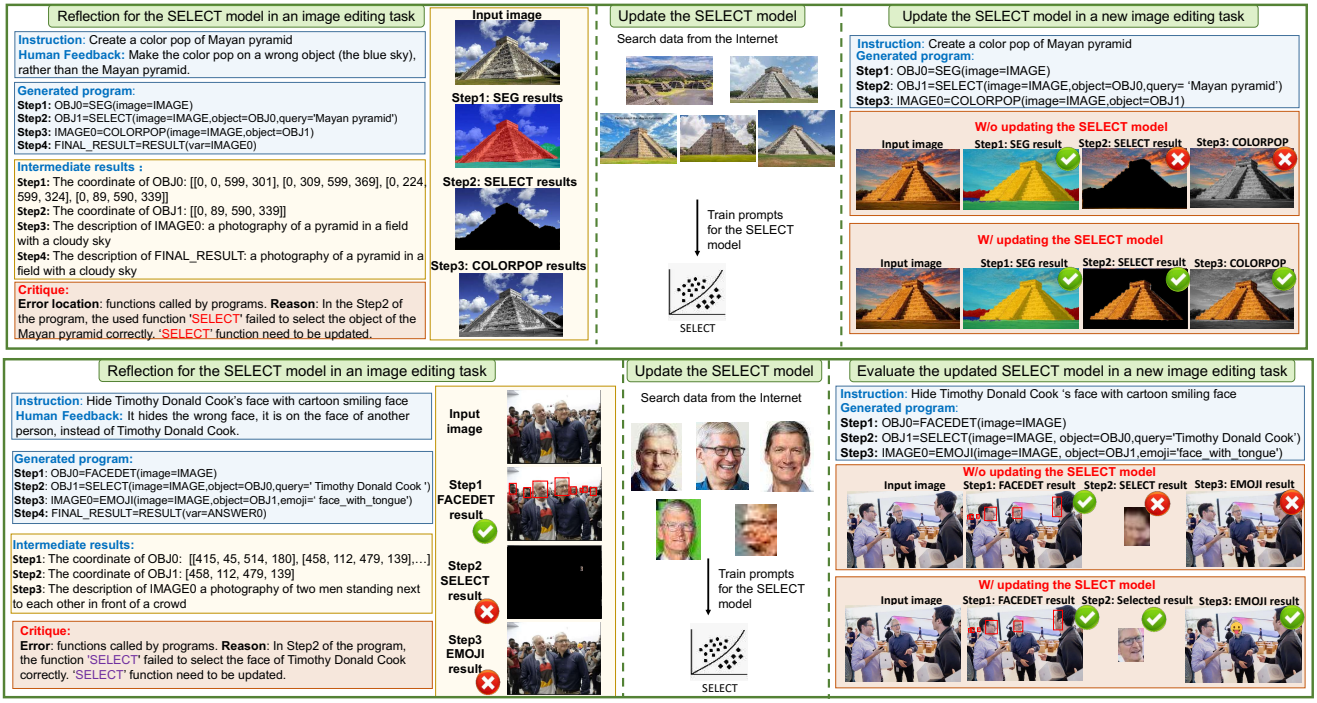

Figure 17. Case studies of updating the SELECT tool.

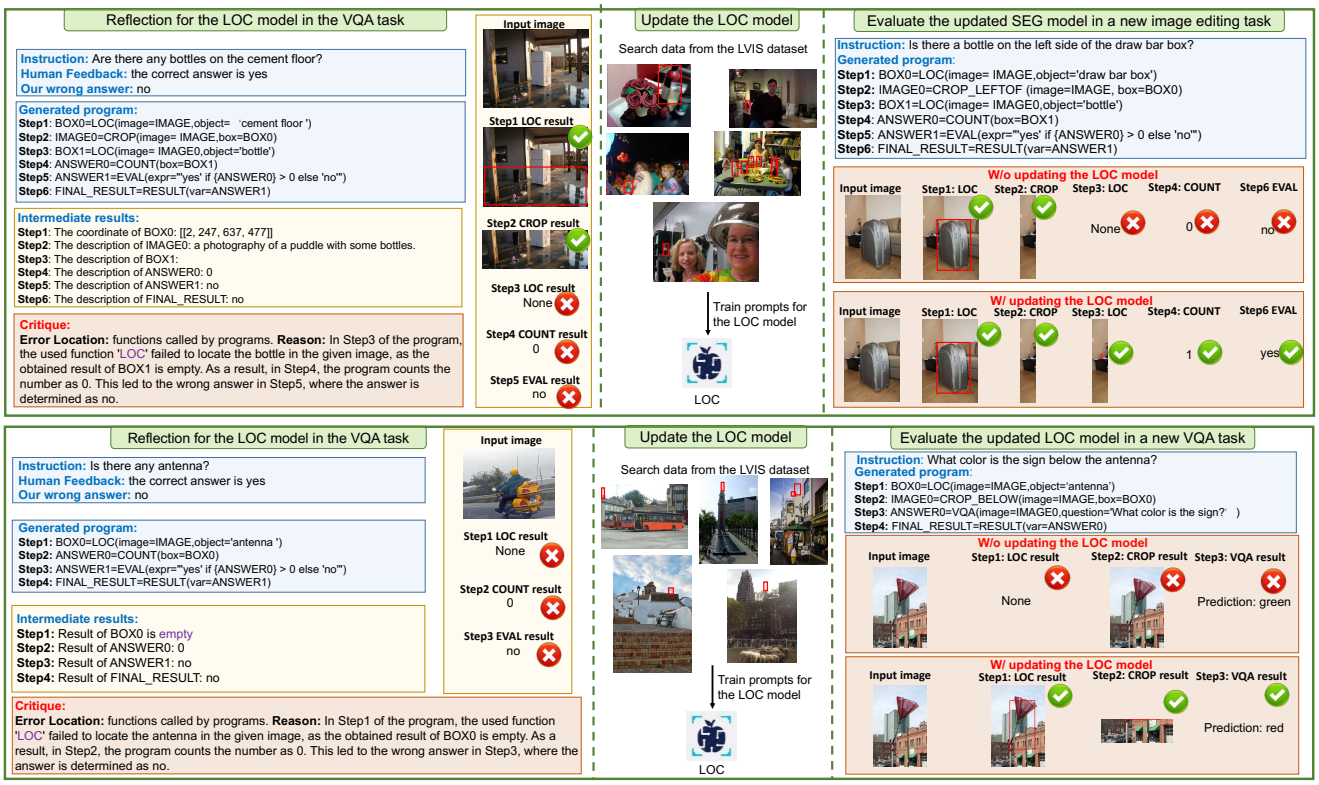

Figure 18. Case studies of updating the LOC tool.

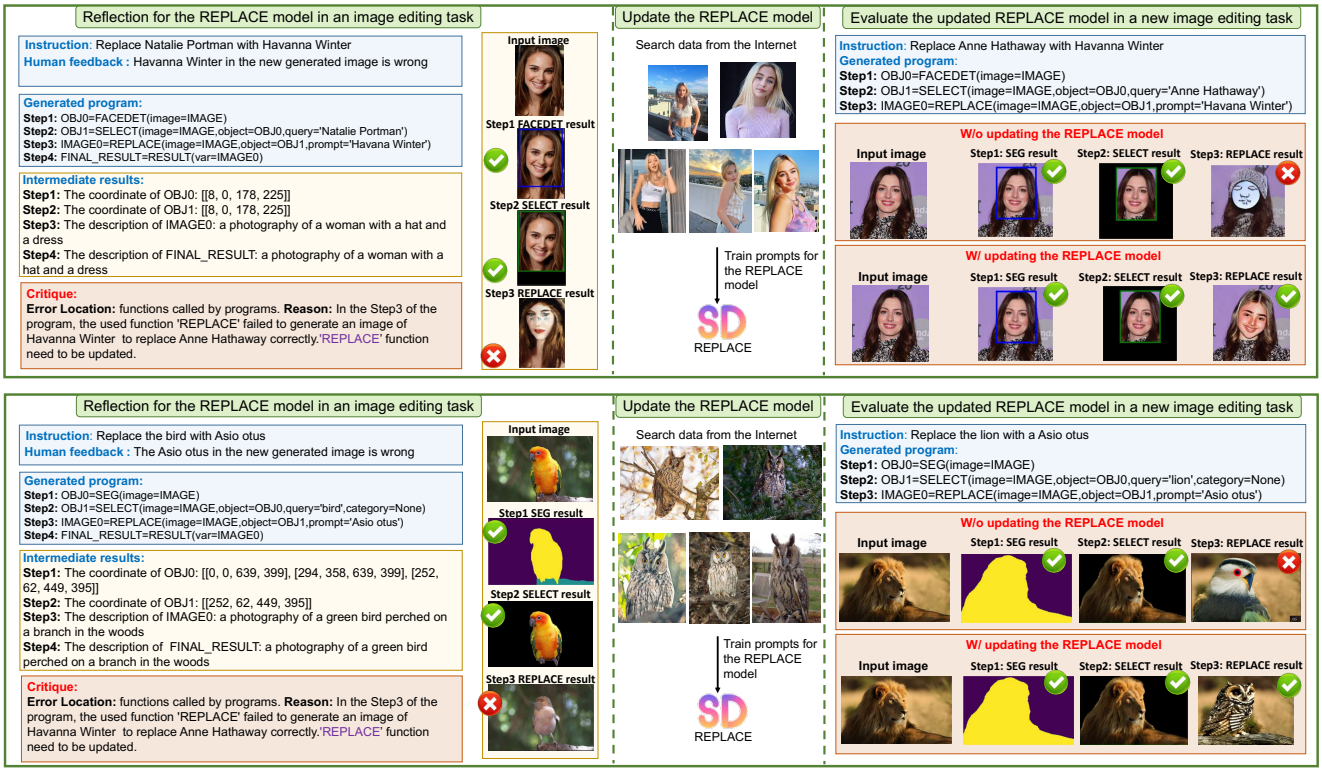

Figure 19. Case studies of updating the REPLACE tool.

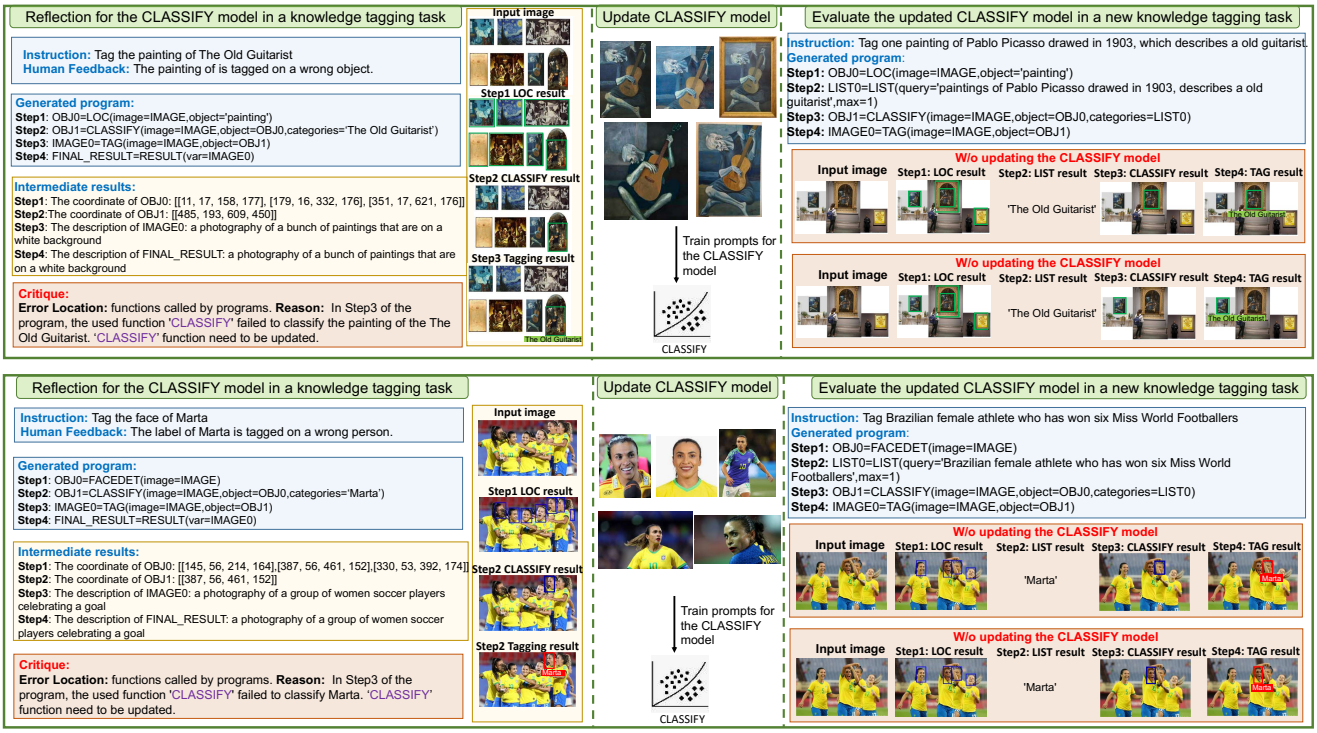

Figure 20. Case studies of updating the CLASSIFY tool.

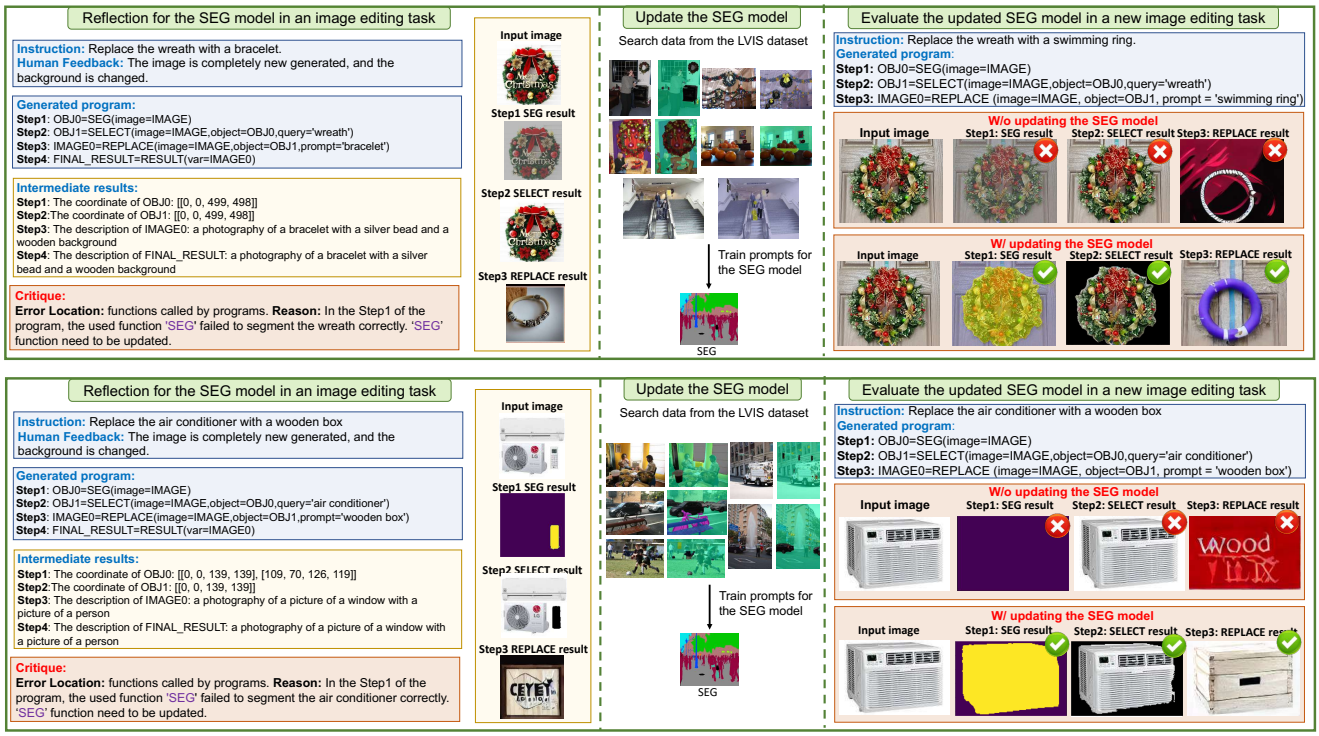

Figure 21. Case studies of updating the SEG tool.

[23] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *ICCV*, pages 11888–11898, 2023.

[24] Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023.

[25] Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. In *NeurIPS*, pages 71995–72007, 2023.

[26] Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023.

[27] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *ICLR*, 2021.