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

Zhi Gao[1,2], Yuntao Du[2], Xintong Zhang[2,3], Xiaojian Ma[2], Wenjuan Han[3], Song-Chun Zhu[1,2,4], Qing Li[2] ✉
[1]Peking University [2]Beijing Institute for General Artificial Intelligence (BIGAI)
[3]Beijing Jiaotong University [4]Tsinghua University
clova-tool.github.io

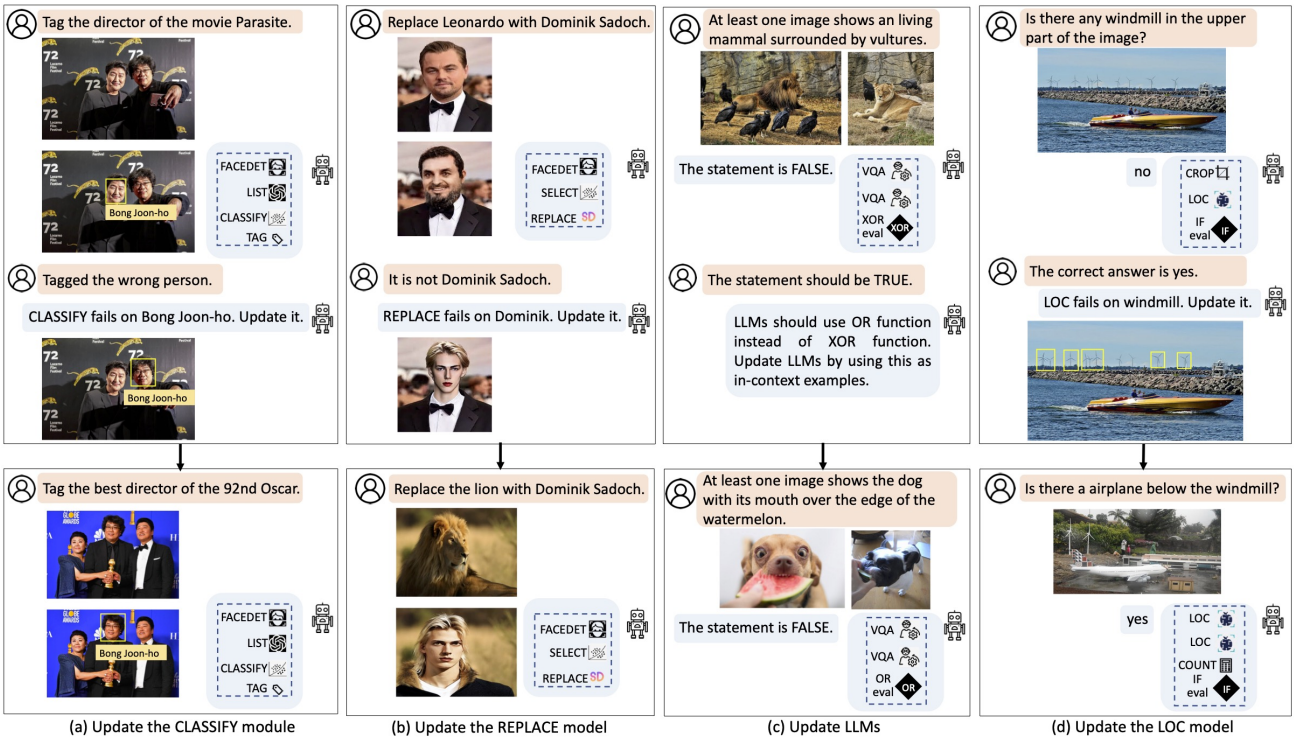

Figure 1. CLOVA is a general visual assistant that updates both LLMs and visual models via inference, reflection, and learning in a closed-loop framework. During inference, CLOVA uses LLMs to integrate visual tools to accomplish given tasks. In reflection, CLOVA identifies models that require updating based on environmental feedback. Finally, in learning, CLOVA collects data and updates models accordingly.

Abstract

Leveraging large language models (LLMs) to integrate off-the-shelf tools (e.g., visual models and image processing functions) is a promising research direction to build powerful visual assistants for solving diverse visual tasks. However, the learning capability is rarely explored in existing methods, as they freeze the used tools after deployment, thereby limiting the generalization to new environments requiring specific knowledge. In this paper, we propose CLOVA, a Closed-LOop Visual Assistant to address this limitation, which encompasses inference, reflection, and learning phases in a closed-loop framework. During inference, LLMs generate programs and execute corresponding tools to accomplish given tasks. The reflection phase introduces a multimodal global-local reflection scheme to analyze whether and which tool needs to be updated based on environmental feedback. Lastly, the learning phase uses three flexible manners to collect training data in real-time and introduces a novel prompt tuning scheme to update the tools, enabling CLOVA to efficiently learn new knowledge without human involvement. Experimental results show that CLOVA outperforms tool-usage methods by 5% in visual question answering and multiple-image reasoning tasks, by 10% in knowledge tagging tasks, and by 20% in image editing tasks, highlighting significance of the learning capability for general visual assistants.

✉ Corresponding author: Qing Li.

1. Introduction

The progresses of large language models (LLMs) [7, 38, 60, 74] and various visual models [6, 26, 36, 47, 48] offer feasible ways to build general visual assistants. Given a task with language instruction, a prevalent way is to leverage LLMs to generate programs, and then execute off-the-shelf visual tools (*e.g.*, OpenCV functions and visual models) to accomplish the task according to the generated program [8, 11, 32, 53, 58, 65, 66]. For example, when asking the question "*What is the person to the left of the umbrella doing?*", a potential solution entails LLMs sequentially (1) using the detection model to locate the umbrella, (2) cropping the image region left to the umbrella, (3) using the detection model to locate the person, and (4) finally querying a visual question answering (VQA) model with "*What is the person doing?*". Due to the remarkable compositionality of this process, tool-usage methods achieve impressive performance in solving complicated visual tasks, such as compositional VQA, vision-and-language navigation, and language-guided image editing [30, 67, 72, 73].

However, the learning capability is rarely explored in existing tool-usage methods. Most of them simply freeze used tools after deployment, limiting the generalization to new environments that require specific knowledge, *e.g.*, how to tackle problems with visual concepts and environments that are novel to the tools. We've demonstrated a few in Fig. 1. For example, the user might instruct the visual assistant to tag the face of the movie director Bong Joon-ho in a picture. However, the face classification tool invoked by the assistant might not be familiar to him, and ultimately the assistant returns with a faulty response. In such cases, we may expect the assistant to be able to learn this missing piece of knowledge on celebrity after the initial failure and generalize it to other tasks, *e.g.* tagging Bong Joon-ho in another picture. Thus, it is crucial to endow visual assistants with the learning capability, enabling them to quickly adapt to new tasks and environments.

In this paper, we propose CLOVA, a Closed-LOop Visual Assistant that updates used tools via closed-loop learning to better adapt to new environments, as shown in Fig. 1. CLOVA comprises three phases: inference, reflection, and learning. During inference, CLOVA employs LLMs to generate programs and execute corresponding tools to accomplish the task at hand. Subsequently in the reflection phase, CLOVA uses environmental feedback to produce critiques, identifying tools that need to be updated. Finally, the learning phase autonomously collects data and updates tools accordingly. In this case, CLOVA enables tools to be updated autonomously, continually improving their ability to adapt to diverse environments.

To establish such a closed-loop learning framework, we must overcome three key challenges. 1) How to identify tools that need to be updated. The multi-step nature of generated programs and the diverse errors in them make identifying the faulty tools challenging. 2) How to collect training data. Real-time collection is necessary, since the knowledge that needs to be learned is unpredictable, causing the difficulty. 3) How to efficiently update tools amid their scale and the quality of collected data. The tools used by the assistant are generally backed by large models and inefficient to be updated, while naive fine-tuning could also lead to unacceptable catastrophic forgetting [34]. The noise within the collected data further perplexes the training issue.

We propose several techniques to tackle these challenges. First, we introduce a multimodal global-local reflection scheme, which resorts to LLMs to identify tools that need to be updated from both global and local aspects. For the second challenge, three data collection manners are employed, including inferring answers by LLMs, searching on the Internet, and searching from open-vocabulary datasets. Lastly, we develop a training-validation prompt tuning scheme for the tools, which includes instance-wise prompt tuning and a subsequent prompt validation stage, where learned prompts that fail to predict the validation data will be discarded. The learning phase also updates LLMs by storing correct examples and incorrect examples with critiques as in-context examples, which will be used in future inference. As a result, CLOVA efficiently updates tools in challenging environments with noisy data, while avoiding existing knowledge being wiped out.

We apply CLOVA to compositional VQA and multiple-image reasoning tasks, using the GQA [17] and NLVRv2 [57] datasets. Additionally, we manually collect data for image editing and factual knowledge tagging tasks. CLOVA outperforms existing tool-usage methods by 5% in compositional visual question answering and multiple-image reasoning tasks, by 10% in knowledge tagging tasks, and by 20% in image editing tasks, showing the significance of the learning capability for general visual assistants.

In summary, our contributions are four-fold. 1) We build CLOVA, a visual assistant that updates its tools within a closed-loop learning framework for better adaptation to new environments. 2) We propose a multimodal global-local reflection scheme, capable of identifying tools in need of updates. 3) We employ three flexible manners to collect training data in real-time, without any human involvement. 4) We introduce a training-validation prompt tuning scheme, enabling models to update themselves in challenging environments efficiently while preserving existing knowledge.

2. Related Work

2.1. General Visual Assistant

Benefiting from the advancements of LLMs [7, 38, 60, 74] and visual models [23, 36, 47, 48], visual assistants have achieved great progresses. Some methods concatenate and

Method	Visual Tool	Reflection	Update LLMs	Update VMs
ART [40]	✗	✗	Prompt	-
TRICE [45]	✗	Global	Instruction + RL	-
ToolkenGPT [12]	✗	-	✗	-
Toolformer [50]	✗	-	Fine-tune	-
VISPROG [11]	✓	✗	✗	✗
Visual ChatGPT [67]	✓	✗	✗	✗
HuggingGPT [53]	✓	✗	✗	✗
ViperGPT [58]	✓	✗	✗	✗
GPT4TOOLs [72]	✓	✗	Instruction	✗
OpenAGI [9]	✓	✗	RL	✗
AssistGPT [8]	✓	Global	Prompt	✗
CLOVA (Ours)	✓	Global+Local	Prompt	Prompt

Table 1. Comparisons with representative tool-usage methods, where VMs means visual models.

train LLMs with visual models in an end-to-end manner, where representative work includes LLaVA [28], Otter [24], MMICL [76], Kosmos-2 [43], and Flamingo [1], *etc*. In addition, some work extends the idea of tool usage for AI assistants from natural language processing [4, 12, 40, 45, 46, 51] to computer vision. By providing in-context examples, VISPROG [11] and ViperGPT [58] generate programs to use visual tools. Following this idea, some work improves performance by collecting instruction-following data [29, 42, 72], adding more tools [27, 53], and designing more dedicated tool-usage procedures [13, 30–32, 67, 68, 73]. The most related work to CLOVA is AssistGPT [8] and OpenAGI [9]. The two methods update LLMs after development through in-context learning and reinforcement learning, respectively. Different from them, CLOVA has the ability to update both LLMs and visual models through its reflection and learning phases. This allows CLOVA to better adapt to new environments. In addition, the closed-loop framework enables us to set a separate training stage, going beyond zero-shot or few-shot visual assistants. Comparisons between CLOVA and some representative tool-usage methods are shown in Tab. 1.

2.2. Reflection of LLMs

Reflection has become a remedy in case LLMs cannot generate good responses in a single attempt [5, 22, 39, 41, 71]. Reflection methods send outputs back to LLMs to obtain critiques and further improve the outputs. These critiques take the form of scores or natural language [35, 37, 55, 79]. To generate better critiques, some methods employ instruction tuning [49, 70] or reinforcement learning [3, 45]. Recently, Huang *et al.* [15] revealed that LLMs struggle to provide accurate critiques for complex tasks. One way to address this issue is incorporating external information such as human-desired results into LLMs [2, 69, 71]. Unlike existing methods that rely solely on feedback in the language modality, our method generates reflection using all multimodal intermediate results. In addition, our method incorporates both global and local aspects for reflection, instead of only the global aspect. These bring more effective critiques for compositional tasks.

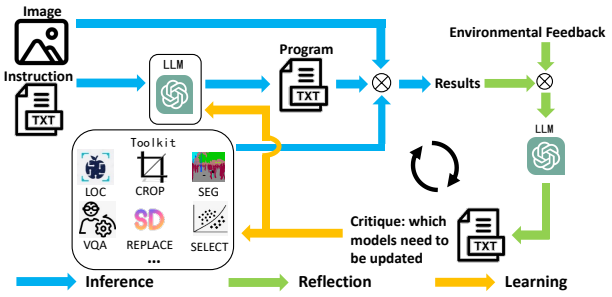

Figure 2. Framework of CLOVA.

2.3. Prompt Tuning

Prompt tuning is an efficient technique to update neural networks, achieving impressive performance in both NLP [19, 54] and computer vision [16, 52, 80, 81]. Hard prompt tuning and soft prompt tuning are two kinds of commonly used methods. Hard prompt tuning develops interpretable tokens (*e.g.*, texts and image regions) to guide model prediction, which are usually obtained by manually designing [47, 59], retrieval [75], and model generation [14, 44]. Soft prompt tuning learns vectors as prompts via gradient-based optimization. VPT [18] and CoOp [78] learn prompts for vision encoders and text encoders, respectively. To handle diverse data, CoCoOp [77] learns to generate prompts for unknown classes, ProDA [33] builds a Gaussian distribution for prompts, and MaPLe [21] uses both the text and visual prompts. In addition, some methods employ soft prompts for continual learning, which learn prompts for different classes and produce adaptive prompts during inference. Representative methods include PIVOT [62], DualPrompt [63], and L2P [64]. Different from existing methods, our training-validation prompt tuning scheme discards harmful prompts, leading to more stable learning processes when the quality of training data is subpar.

3. Method

3.1. Overview

CLOVA has three phases: inference, reflection, and learning, as shown in Fig. 2. In the inference phase, CLOVA uses LLMs to generate programs and executes corresponding tools to solve the task. The reflection phase introduces a multimodal global-local reflection scheme that uses LLMs to generate critiques, identifying which tool needs to be updated. During learning, we employ three manners to collect training data and use a training-validation prompt tuning scheme to update the tools.

3.2. Inference

Our inference phase is based on VISPROG [11], while the difference is that CLOVA first uses LLMs to generate plans and then generates programs based on the plans, instead of

Tool Type	Tool Name	Tool Description	Data Collection
Tools to be updated	LOC	Use the OWL-ViT model [36] for object localization	Open-vocabulary datasest
	VQA	Use the BLIP model [26] for VQA	LLM inference
	SEG	Use the maskformer model [6] for panoptic segmentation	Open-vocabulary datasest
	SELECT	Use the CLIP model [47] to select the most relevant object, given a text description	Internet
	CLASSIFY	Use the CLIP model [47] to classify given images	Internet
	REPLACE	Use the stable diffusion inpainting model [48] to replace one object with another desirable object	Internet
Tools not to be updated	FACEDET	Use the DSFD model [25] for face detection	N/A
	LIST	Use the text-davinci-002 model of OpenAI for knowledge retrieval	N/A
	EVAL	Use the Python function eval() to process string expressions for answers	N/A
	RESULT	Use the Python function dict() to output the intermediate and final results	N/A
	COUNT	Use Python function len() to count the number of input bounding boxes or masks	N/A
	CROP	Use Python function PIL.crop() to crop images	N/A
	COLORPOP	Use Python function PIL.convert() to keep desirable objects in color and other regions gray	N/A
	BGBLUR	Use Python function PIL.GaussianBlur() to blur the background	N/A
	EMOJI	Use emojis in the Python packages AngLy(pypi) to hide someone's face	N/A

Table 2. Used tools in CLOVA, categorized based on whether the tool is updated in our method or not. Details on tool updates is in Sec. 3.4

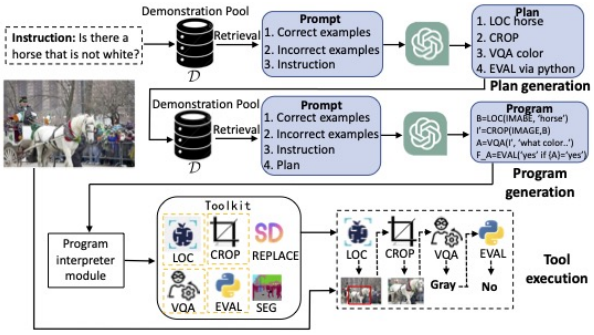

Figure 3. Illustration of the inference phase in CLOVA.

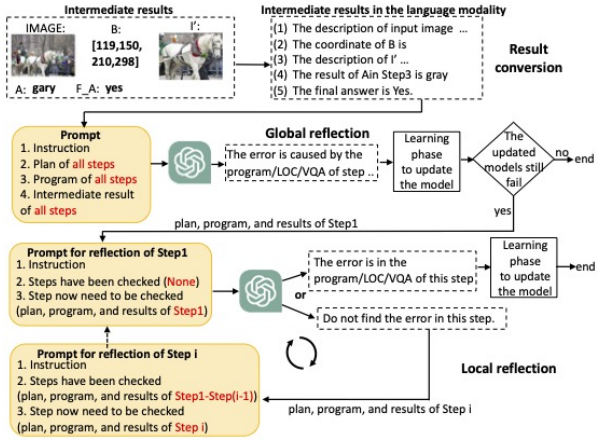

Figure 4. Illustration of the reflection phase in CLOVA.

directly generating programs. Plans can be seen as intermediate reasoning chains that benefit the inference and reflection phases. Given a task, CLOVA selects in-context examples from a demonstration pool $\mathcal{D}$ (the construction of $\mathcal{D}$ will be detailed in Sec. 3.4.1), including correct examples and incorrect examples with error critiques. These examples are used to create prompts that are then sent to LLMs for plan and program generation. Finally, the program is parsed to execute visual tools (see Fig. 3).

Plan generation. $\mathcal{D}$ is composed by $\mathcal{D} = \{\mathcal{D}_{p,s}, \mathcal{D}_{p,f}, \mathcal{D}_{c,s}, \mathcal{D}_{c,f}\}$, where $\mathcal{D}_{p,s}$ and $\mathcal{D}_{p,f}$ contain correct and incorrect examples for plan generation respectively, and $\mathcal{D}_{c,s}$ and $\mathcal{D}_{c,f}$ contain correct and incorrect examples for program generations respectively. Given a task, we use the BERT model [20] to extract features of the given instruction and examples stored in $\mathcal{D}_{p,s}$ and $\mathcal{D}_{p,f}$. Then, we combine similar examples in $\mathcal{D}_{p,s}$ and $\mathcal{D}_{p,f}$ with the instruction to create a prompt. Finally, we send the prompt to LLMs to generate the plan in a one-go manner.

Program generation. Similar to plan generation, we use LLMs to generate programs in a one-go manner. We select correct and incorrect examples of programs from $\mathcal{D}_{c,s}$ and $\mathcal{D}_{c,f}$. These examples are then combined with the plan as a prompt that is sent to LLMs for program generation.

Tool execution. We utilize the interpreter module in [11] to parse the program, extracting tool names, inputs, and outputs of each step. CLOVR activates tools from a toolkit $\mathcal{T}$ that contains 15 tools totally, including visual models and Python functions, as shown in Tab. 2.

3.3. Reflection

In the inference phase, if a task is not solved correctly, the multimodal global-local reflection scheme uses LLMs to generate critiques, identifying which tool needs to be updated, as shown in Fig. 4.

Result conversion. Since LLMs often struggle to identify errors by themselves [15, 56, 61], we provide the environmental feedback, desirable results, our wrong results, and intermediate results of each step for LLMs to better identify the error source. This requires us to convert visual results into textual form. For this purpose, we use the BLIP [26] model to convert images into languages by captioning.

Global reflection. CLOVA first uses global reflection to generate critiques in a one-go manner. The prompts are composed of task inputs, feedback on the task (*e.g.*, desirable results in VQA tasks, or human comments on image editing tasks), generated plan and program, and intermediate results at each step. We send the prompts to LLMs to generate critiques that are used to update tools in the learning phase.

Local reflection. If CLOVA still fails after the tools are updated via global reflection and the learning phase –meaning the actual tools that lead to the faulty response are still to be found, we resort to local reflection to analyze each step of the program. Prompts are composed of the task inputs, feedback on the task, the steps that have been checked, and the current step that needs to be checked. Each step includes plans, programs, and intermediate results. We send the prompts to LLMs to infer whether this step has errors and reasons. Local reflection continues until an error location and reasons are identified for a step.

3.4. Learning

After obtaining tools that need to be updated from the reflection phase, CLOVA then moves to the learning phase to collect training data and goes through training-validation prompt tuning to update them, as shown in Fig. 5. **Data collection.** Since the tools that need to be updated can be rather different (a full list can be found in Tab. 2), we explore three manners to collect data in real-time: 1) We use LLMs to generate training data for the VQA tool. If reflection concludes that the VQA model makes errors, we combine the desirable response of the whole task and intermediate results of each step to prompt LLMs into inferring the correct output of the VQA tool. The question and the inferred output are then used to update the VQA tool. 2) We gather training data from open-vocabulary visual object grounding datasets (*e.g.*, LVIS [10]) for the LOC and SEG tools. For example, if the reflection phase indicates that LOC does not work well for "*horse*", CLOVA will select images and bounding boxes of horses from LVIS to update LOC. 3) We collect data by searching on the Internet for SELECT, CLASSIFY, and REPLACE tools. For instance, in case CLASSIFY is flagged by the reflection on not recognizing "*horse*" well, CLOVA will search for images of horses on the Internet to update CLASSIFY.

Prompt training and validation. Given the collected data, we invoke training-validation prompt tuning to update tools. Note that, instead of training a single soft prompt for all the collected data, we choose to learn a soft prompt for each training instance collected. Each learned prompt will then be validated by running the tool with it on validation data (held out from collected data except for the VQA tool, where VQA will be validated on the original visual question it failed on) and seeing if the tool can produce the response we expect in an update (*e.g.* correctly localizing a horse for the LOC tool). As a result, we can discard prompts that do not lead to the expected tool update possibly due to the faulty training instances they were trained on, alleviating the issue of possibly noisy collected data. Finally, we build a prompt pool $\mathcal{P}$ **for each tool**. Taking the LOC tool as an example: after training and validation, CLOVA stores the visual concept (*e.g.*, "horse" in the task of localizing horses), the learned prompts and the visual features of all collected instances in $\mathcal{P}$, formulated as $\mathcal{P} = \{v_j : [[f_{j1}, \cdots, f_{jn}], [p_{j1}, \cdots, p_{jn}]]\}_{j=1}^{m}$ where v_j is the name of the j-the concept, m concepts are stored totally. For concept v_j, f_{ji} and p_{ji} are the feature and learned prompt for the i-th instance, respectively, and n instances are learned for v_j.

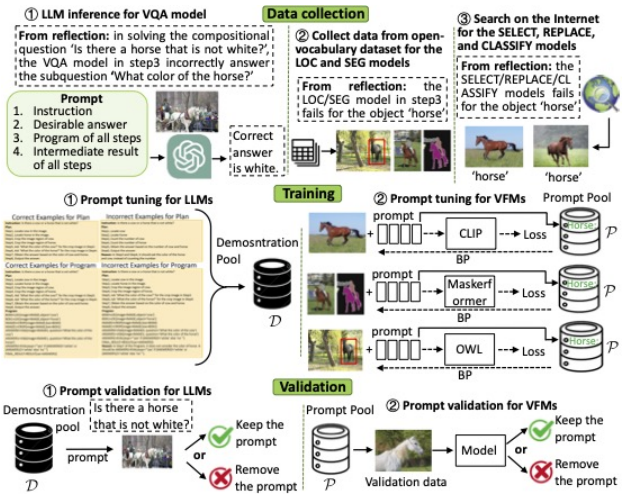

Figure 5. Illustration of the learning phase in CLOVA.

Prompt ensemble. During inference, we use prompt ensemble to retrieve and utilize prompts from the aforementioned prompt pool $\mathcal{P}$. Specifically, given an instruction and the generated program, we first identify the visual concept of each tool being used. For example, given an image editing task, "*Replace the dog with a cat*" where SEG, SELECT, and REPLACE models will be used, "*dog*" is the visual concept for SEG and SELECT, and "*cat*" is the visual concept for REPLACE. Then, in each step of tool-use with an input image x, if the visual concept of the tool-use is not in $\mathcal{P}$ of the corresponding tool, the soft prompt p' will be set as a zero-vector, *i.e.* the tool has not been updated with this concept so we just use the original tool; if it can be found in $\mathcal{P}$, *i.e.* the tool was updated with v_j before, we aggregate the soft prompt corresponding to this concept based on the similarity between the visual features stored with these prompts and feature extracted from the current input image x: p' is computed by $p' = \frac{\sum_{i=1}^{n} w_i \cdot p_{ji}}{\sum_{i=1}^{n} w_i}$, where we compute the cosine similarity between feature f_x of x and features f_{ji} in $\mathcal{P}$ as the weight w_i.

3.4.1 LLM learning with demonstrations

Besides the tools, CLOVA can also update its LLMs through in-context learning. As we mentioned above, CLOVA utilizes a demonstration pool $\mathcal{D}$ to provide relevant examples for the LLMs. After working on a new problem, the plan, program, and reflection will be stored into $\mathcal{D}$ based

	Method	GQA	NLVRv2	Image Editing	Tagging
End-to-end	Otter [24]	48.2	48.2	-	-
	MMICL [76]	64.4	62.2	-	-
Tool-usage	GPT4TOOLs [67]	41.2	45.4	17.8	
	Visual ChatGPT [67]	43.2	51.6	21.7	-
	InternGPT [30]	44.8	39.4	-	-
	HuggingGPT [53]	46.0	44.0	-	-
	ViperGPT [58]	47.2	-	-	-
	VISPROG [11]	49.8	60.8	40.2	39.3
	CLOVA (Ours)	**54.6**	**65.6**	**65.4**	**50.2**

Table 3. Comparisons in the four tasks. We report accuracies on GQA, NLVRv2, and image editing tasks, and F1 score on the tagging task.

on whether this problem is correctly solved (then it will be marked as a correct example) or the opposite (as an incorrect example). We also have a validation process that uses the original instruction as validation data to evaluate stored in-context examples. As the size of $\mathcal{D}$ grows, LLMs use more examples and therefore strengthen reasoning skills.

The framework of CLOVA is summarized in Algorithm 1.

Algorithm 1 CLOVA

Input: LLMs, visual models, 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 soft prompts in $\mathcal{P}$.
    if Updated models solve T_i incorrectly **then**
      Perform local reflection.
      Perform training-validation prompt tuning, store in-context examples into $\mathcal{D}$ and soft prompts in $\mathcal{P}$.
    end if
  end if
end for

4. Experiments

4.1. Setting

Following VISPROG [11], we evaluate our method on four tasks: compositional VQA, multiple-image reasoning, language-guided image editing, and factual knowledge tagging, which requires visual perception, compositional reasoning, and image generation and manipulation abilities.

To comprehensively evaluate the learning ability of CLOVA, we set a separate training stage before deployment, which iteratively learns knowledge via inference, reflection, and updating phases. In the test stage (*i.e.*, after development), we do not update LLMs and visual models, and evaluate the performance only via the inference phase.

In the compositional VQA task, the GQA dataset [17] is used. We randomly select 500 samples from its train split as the training data, and 500 samples from its test-dev split as the test data. We report the top-1 accuracy. In the multiple-image reasoning task, we use the NLVRv2 dataset [57] that provides two images and a statement. We need to judge whether the statement is true or false. Similarly, we randomly select 500 samples from its train split as our training data, and 500 samples from its dev split as our test data.

Similar to VISPROG, we manually collect data for the language-guided image editing and factual knowledge tagging tasks. To better evaluate the learning capability, we collect fine-grained visual concepts that visual models may not have learned, such as "*Replace the lion in the image with pine grosbeak*", where pine grosbeak is a fine-grained bird of Passeriformes. In the image editing task, we collect 129 images with 193 instructions, where 27 images with 78 instructions are used for training, and the rest are test data. We manually check whether edited images are semantically correct. The factual knowledge tagging task needs the model to identify persons or objects with bounding boxes in images. We collect 86 images with 177 instructions for this task, where 10 images with 88 instructions are used for training and the rest are used as the test data. We report the F1 score for this task.

In plan and program generation, prompts contain 4 correct examples and 4 incorrect examples. The demonstration pool $\mathcal{D}$ is initialized having about 15 in-context examples.

4.2. Main Results

We compare CLOVA with tool-usage methods: VISPROG [11], GPT4TOOLs [72], Visual ChatGPT [67], InternGPT [30], HuggingGPT [53], and ViperGPT [58]. We use their official codes, where all methods use the GPT-3.5 model. In addition, we also compare CLOVA with two advanced end-to-end models: Otter [24] and MMICL [76], which do well in GQA and NLVRv2 datasets. Results on the four tasks are shown in Tab. 3. We observe that CLOVA achieves the best performance among tool-usage

Figure 6. Case study of CLOVA on four example tasks.

methods. CLOVA has at least 4.8% improvements on the GQA dataset and 4.9% improvements on the NLVRv2 dataset. The reason is that CLOVA learns how to generate programs for the two tasks and update the VQA and LOC models for better image perception. CLOVA performs competitively and even outperforms Otter and MMICL.

On image editing and knowledge tagging tasks, we do not compare our method with InterGPT, HuggingGPT, and ViperGPT, since they either need to manually input object masks or cannot accurately locate fine-grained objects. In addition, most methods cannot finish tagging tasks. Thus, we compare our method with VISPROG and Visual ChatGPT. As we collect find-grained data, it is challenging for off-the-shelf classification, segmentation, and image generation models. Since GPT4TOOLs and Visual ChatGPT cannot use OpenCV functions and do not have the learning

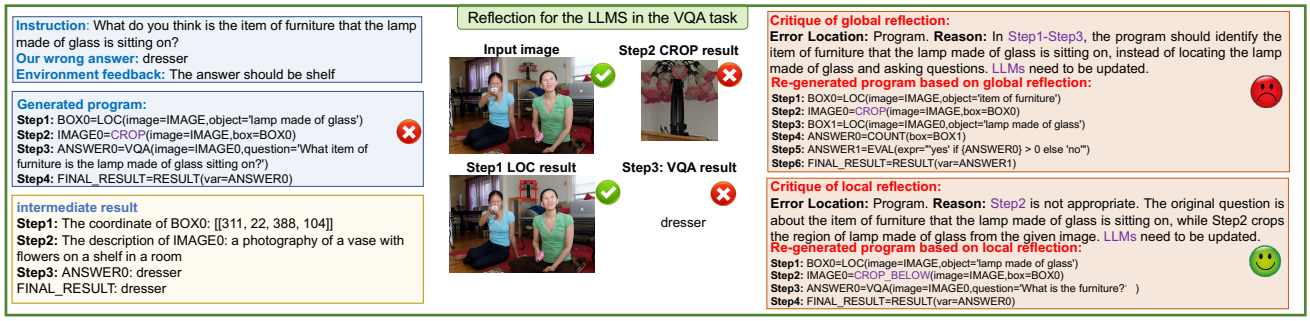

Figure 7. Visualization of the global reflection and local reflection in a VQA task.

	Method	GQA	NLVRv2
Reflection	w/o local reflection	52.0	65.2
	w/o global reflection	53.6	64.2
	w/o intermediate results	48.8	61.2
	w/o plan	50.0	62.6
	Ours	**54.6**	**65.6**
Prompt Tuning for LLMs	w/o incorrect cases	46.1	61.4
	w/o correct cases	48.2	63.2
	w/o validation	44.2	61.0
	Ours	**54.6**	**65.6**
Prompt Tuning for visual models	w/o validation	42.8	62.8
	Ours	**54.6**	**65.6**

Table 4. Ablation on the GQA and NLVRv2 dataset.

capability, they get bad performance on the image editing task. VISPROG can use OpenCV functions, but it cannot learn new knowledge. Its main fault is the inability to recognize or generate fine-grained concepts. Compared with them, the learning ability of CLOVA brings more than 20% improvements in image editing tasks, and 10% improvements in knowledge tagging tasks.

4.3. Case Study

In Fig. 6, we visualize four cases to illustrate the reflection and learning ability in CLOVA. It is capable of identifying models that need to be updated, no matter LLMs or visual models. Hard prompts guide LLMs to generate correct programs for similar instructions. Visual models could learn new concepts via the proposed data collection manners and soft prompt turning scheme. In Fig. 7, we visualize an example of global and local reflection. When the instruction is complex, the global reflection does not accurately identify which step has the error. Using global reflection as in-context examples still cannot generate correct programs. In contrast, local reflection successfully identifies the error step and using local reflection generates correct programs.

4.4. Ablation

We conduct ablations to evaluate the reflection and learning phases, using the GQA and NLVRv2 datasets. For reflection, we evaluate only using global reflection, only using local reflection, not using multimodal intermediate results, and not generating plans. We separately evaluate hard and soft prompts for learning. We evaluate only storing correct examples and only storing incorrect examples for hard prompt tuning. We also evaluate removing the validation process in hard and soft prompt tuning processes. Results are shown in Tab. 4. We find that these components are necessary for CLOVA to achieve better performance.

Dataset	Method	LLama2-7B	GPT-3.5-turbo	GPT-4
GQA	Baseline	39.2	46.4	52.6
	+ Update LLMs	56.8	51.6	56.6
	+ Update visual models	60.2	54.6	60.4
NLVRv2	Baseline	50.0	60.2	64.8
	+ Update LLMs	59.2	63.6	68.8
	+ Update visual models	63.8	65.6	69.2

Table 5. Different LLMs on the GQA and NLVRv2 datasets.

We evaluate our method using different LLMs: Llama2-7B, GPT-3.5-turbo, and GPT-4. The versions GPT-3.5-turbo and GPT-4 are in October 2023. Results on GQA and NLVRv2 are shown in Tab. 5. We find that on both two datasets with three LLMs, updating LLMs and visual models indeed leads to improvements. In addition, we observe that our method achieves higher improvements on open-source LLMs (*i.e.*, Llama2-7B), 21% on the GQA dataset and 13.8% on the NLVRv2 dataset, bringing significance of studying the learning capability of visual assistants.

5. Conclusion

In this paper, we have presented CLOVA, a general visual assistant that can adapt to new environments via inference, reflection, and learning in a closed-loop framework. In the inference phase, using both correct and incorrect examples for prompts benefits to generate better plans and programs. Our reflection scheme is capable of identifying tools that need to be updated. Through three flexible data collection manners and the validation-learning prompt tuning scheme in the learning phase, CLOVA can efficiently improve its tools while avoiding catastrophic forgetting. Experimental results on four tasks show the effectiveness of CLOVA as a general visual assistant with learning abilities. In the current method, we assume there is no selection or loop structure in programs, and assume there is at most one model that needs updates in a task. The two assumptions cannot always hold in real world. We are going to break the assumptions, making the closed-loop framework more practical.

References

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *NeurIPS*, pages 23716–23736, 2022.

[2] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv*, abs/2310.11511, 2023.

[3] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[4] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *ArXiv*, abs/2305.17126, 2023.

[5] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

[6] Bowen Cheng, Alex Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. *NeurIPS*, 34:17864–17875, 2021.

[7] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[8] 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.

[9] Yingqiang Ge, Wenyue Hua, Jianchao Ji, Juntao Tan, Shuyuan Xu, and Yongfeng Zhang. Openagi: When llm meets domain experts. 2023.

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

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

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

[13] 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.

[14] Xiaowei Hu, Zhe Gan, Jianfeng Wang, Zhengyuan Yang, Zicheng Liu, Yumao Lu, and Lijuan Wang. Scaling up vision-language pre-training for image captioning. In *CVPR*, pages 17980–17989, 2022.

[15] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

[16] Siteng Huang, Biao Gong, Yulin Pan, Jianwen Jiang, Yiliang Lv, Yuyuan Li, and Donglin Wang. Vop: Text-video co-operative prompt tuning for cross-modal retrieval. In *CVPR*, pages 6565–6574, 2023.

[17] Drew A. Hudson and Christopher D. Manning. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *CVPR*, pages 6700–6709, 2019.

[18] 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.

[19] Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8: 423–438, 2020.

[20] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186, 2019.

[21] Muhammad Uzair Khattak, Hanoona Rasheed, Muhammad Maaz, Salman Khan, and Fahad Shahbaz Khan. Maple: Multi-modal prompt learning. In *CVPR*, pages 19113–19122, 2023.

[22] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *arXiv preprint arXiv:2303.17491*, 2023.

[23] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollar, and Ross Girshick. Segment anything. In *ICCV*, pages 4015–4026, 2023.

[24] Bo Li, Yuanhan Zhang, Liangyu Chen, Jinghao Wang, Jingkang Yang, and Ziwei Liu. Otter: A multi-modal model with in-context instruction tuning. *arXiv preprint arXiv:2305.03726*, 2023.

[25] Jian Li, Yabiao Wang, Changan Wang, Ying Tai, Jianjun Qian, Jian Yang, Chengjie Wang, Jilin Li, and Feiyue Huang. Dsfd: dual shot face detector. In *CVPR*, pages 5060–5069, 2019.

[26] 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.

[27] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*, 2023.

[28] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023.

[29] 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.

[30] 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.

[31] 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.

[32] 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*, 2023.

[33] Yuning Lu, Jianzhuang Liu, Yonggang Zhang, Yajing Liu, and Xinmei Tian. Prompt distribution learning. In *CVPR*, pages 5206–5215, 2022.

[34] Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yuechen Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *ArXiv*, abs/2308.08747, 2023.

[35] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

[36] Matthias Minderer, Alexey Gritsenko, Austin Stone, Maxim Neumann, Dirk Weissenborn, Alexey Dosovitskiy, Aravindh Mahendran, Anurag Arnab, Mostafa Dehghani, Zhuoran Shen, et al. Simple open-vocabulary object detection. In *ECCV*, pages 728–755, 2022.

[37] Varun Nair, Elliot Schumacher, Geoffrey Tso, and Anitha Kannan. Dera: enhancing large language model completions with dialog-enabled resolving agents. *arXiv preprint arXiv:2303.17071*, 2023.

[38] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[39] Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. *arXiv preprint arXiv:2308.03188*, 2023.

[40] 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.

[41] Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023.

[42] 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.

[43] Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, and Furu Wei. Kosmos-2: Grounding multimodal large language models to the world. *arXiv preprint arXiv:2306.14824*, 2023.

[44] Sarah Pratt, Ian Covert, Rosanne Liu, and Ali Farhadi. What does a platypus look like? generating customized prompts for zero-shot image classification. In *ICCV*, pages 15691–15701, 2023.

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

[46] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*, 2023.

[47] 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.

[48] 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.

[49] William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*, 2022.

[50] 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*, 2023.

[51] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

[52] Zhenwei Shao, Zhou Yu, Meng Wang, and Jun Yu. Prompting large language models with answer heuristics for knowledge-based visual question answering. In *CVPR*, pages 14974–14983, 2023.

[53] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*, 2023.

[54] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *EMNLP*, pages 4222–4235, 2020.

[55] Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

[56] Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. Gpt-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems. *arXiv preprint arXiv:2310.12397*, 2023.

[57] Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. A corpus for reasoning about natural language grounded in photographs. In *ACL*, pages 6418–6428, 2019.

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

[59] Jiajin Tang, Ge Zheng, Jingyi Yu, and Sibei Yang. Cotdet: Affordance knowledge prompting for task driven object detection. In *ICCV*, pages 3068–3078, 2023.

[60] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[61] Karthik Valmeekam, Matthew Marquez, and Subbarao Kambhampati. Can large language models really improve by self-critiquing their own plans? *arXiv preprint arXiv:2310.08118*, 2023.

[62] Andrés Villa, Juan León Alcázar, Motasem Alfarra, Kumail Alhamoud, Julio Hurtado, Fabian Caba Heilbron, Alvaro Soto, and Bernard Ghanem. Pivot: Prompting for video continual learning. In *CVPR*, pages 24214–24223, 2023.

[63] Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *ECCV*, pages 631–648, 2022.

[64] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *CVPR*, pages 139–149, 2022.

[65] Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *arXiv preprint arXiv: 2311.05997*, 2023.

[66] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multitask agents. *arXiv preprint arXiv:2302.01560*, 2023.

[67] 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.

[68] Jinjin Xu, Liwu Xu, Yuzhe Yang, Xiang Li, Yanchun Xie, Yi-Jie Huang, and Yaqian Li. u-llava: Unifying multi-modal tasks via large language model. *2311.05348,arXiv*, 2023.

[69] Shicheng Xu, Liang Pang, Huawei Shen, Xueqi Cheng, and Tat-seng Chua. Search-in-the-chain: Towards the accurate, credible and traceable content generation for complex knowledge-intensive tasks. *arXiv preprint arXiv:2304.14732*, 2023.

[70] Hao Yan, Saurabh Srivastava, Yintao Tai, Sida I Wang, Wentau Yih, and Ziyu Yao. Learning to simulate natural language feedback for interactive semantic parsing. *arXiv preprint arXiv:2305.08195*, 2023.

[71] Kevin Yang, Yuandong Tian, Nanyun Peng, and Dan Klein. Re3: Generating longer stories with recursive reprompting and revision. In *EMNLP*, pages 4393–4479, 2022.

[72] 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. *arXiv preprint arXiv:2305.18752*, 2023.

[73] 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.

[74] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. Glm-130b: An open bilingual pre-trained model. In *ICLR*, 2022.

[75] Yuanhan Zhang, Kaiyang Zhou, and Ziwei Liu. What makes good examples for visual in-context learning? *arXiv preprint arXiv:2301.13670*, 2023.

[76] Haozhe Zhao, Zefan Cai, Shuzheng Si, Xiaojian Ma, Kaikai An, Liang Chen, Zixuan Liu, Sheng Wang, Wenjuan Han, and Baobao Chang. Mmicl: Empowering vision-language model with multi-modal in-context learning. *arXiv preprint arXiv:2309.07915*, 2023.

[77] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Conditional prompt learning for vision-language models. In *CVPR*, pages 16816–16825, 2022.

[78] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Learning to prompt for vision-language models. *IJCV*, 130(9):2337–2348, 2022.

[79] Pei Zhou, Aman Madaan, Srividya Pranavi Potharaju, Aditya Gupta, Kevin R. McKee, Ari Holtzman, Jay Pujara, Xiang Ren, Swaroop Mishra, Aida Nematzadeh, Shyam Upadhyay, and Manaal Faruqui. How far are large language models from agents with theory-of-mind? *arXiv*, abs/2310.03051, 2023.

[80] Jiawen Zhu, Simiao Lai, Xin Chen, Dong Wang, and Huchuan Lu. Visual prompt multi-modal tracking. In *CVPR*, pages 9516–9526, 2023.

[81] Muzhi Zhu, Hengtao Li, Hao Chen, Chengxiang Fan, Weian Mao, Chenchen Jing, Yifan Liu, and Chunhua Shen. Segprompt: Boosting open-world segmentation via category-level prompt learning. In *ICCV*, pages 999–1008, 2023.

[82] 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.

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

Supplementary Material

List of Figures

List of Tables

6. Comparisons with related methods

In Tab. 6, 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 ability, and few methods use global reflection to improve the plan or programs generated by LLMs. Different from them, CLOVA uses both global reflection and local reflection to identify models 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.

7. Prompt Examples

7.1. 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. 8 to 11.

7.2. Prompts in inference

In the inference phase, we use LLMs to generate plans and programs. We show examples of prompts for plan gener-

(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 8. 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 [40]	✗	✗	Prompt	-
LATM [4]	✗	Global	Prompt	-
TRICE [45]	✗	Global	Instruction + RL	-
ToolkenGPT [12]	✗	-	✗	-
Toolformer [50]	✗	-	Fine-tune	-
VISPROG [11]	✓	✗	✗	✗
Visual ChatGPT [67]	✓	✗	✗	✗
InternGPT [30]	✓	✗	✗	✗
HuggingGPT [53]	✓	✗	✗	✗
ViperGPT [58]	✓	✗	✗	✗
ToT [13]	✓	✗	✗	✗
Chameleon [32]	✓	✗	✗	✗
ControlLLM [31]	✓	✗	✗	✗
MM-REACT [73]	✓	✗	✗	✗
Llava-plus [29]	✓	✗	Instruction	✗
Gorilla [42]	✓	✗	Instruction	✗
GPT4TOOLs [72]	✓	✗	Instruction	✗
OpenAGI [9]	✓	✗	Reinforcement Learning	✗
AssistGPT [8]	✓	Global	Prompt	✗
CLOVA (Ours)	✓	Global+Local	Prompt	Prompt

Table 6. Comparisons with representative tool-usage methods, where VMs means visual models.

ation and program generation in Figs. 12 and 13, respectively.

7.3. 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. 14 and 15, respectively.

7.4. Prompts in learning

In the learning phase, we use LLMs to infer answers for the VQA model, and then train the VQA model using the question and inferred answer. One example of prompts sent to LLMs for answer inferring is shown in Fig. 16, respectively.

8. Details of Tool update

8.1. Update VQA tool

8.1.1 Model

We use the BLIP [26] 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. 17.

8.1.2 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. 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 [26] 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.

8.1.3 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.

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

8.2. Update LOC tool

8.2.1 Model

In CLOVA, we use the OWL-ViT model [22] for object localization as the LOC tool. The architecture of OWL-ViT is shown in Fig. 18. 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.

8.2.2 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 [10] for OWL-

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

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 [82] while the latter uses L_1 loss. For classification loss, we regard the 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

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 12. 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 13. 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
Environmental 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
Environmental 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 14. Prompts for global reflection.

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.

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 15. Prompts for local reflection.

8.2.3 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 instance and contact the aa'gag'gaggrag'g'reaggregaggregaaggrega prompt with the input to produce a prediction.

8.3. Update SEG tool

8.3.1 Model

We use the Maskformer [6] 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 decoder. The architecture of Maskformer is shown in Fig. 19.

8.3.2 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 256. We remove the classification loss of Maskformer, and only use the mask loss [6] 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 [10]. 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.

8.3.3 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.

8.4. Update SELECT and CLASSIFY tools

8.4.1 Model

We utilize CLIP [47] 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

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
Environmental 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 16. Prompts of inferring answers for the VQA model.

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 im-

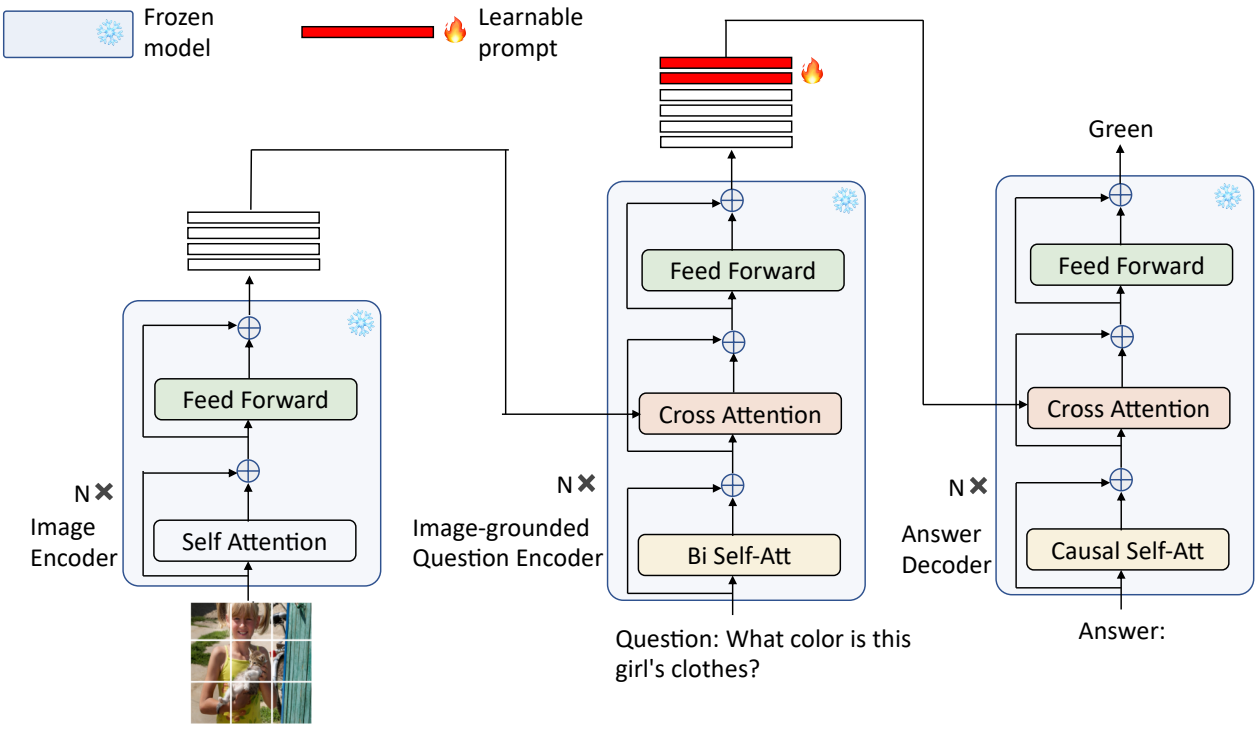

Figure 17. The architecture of BLIP.

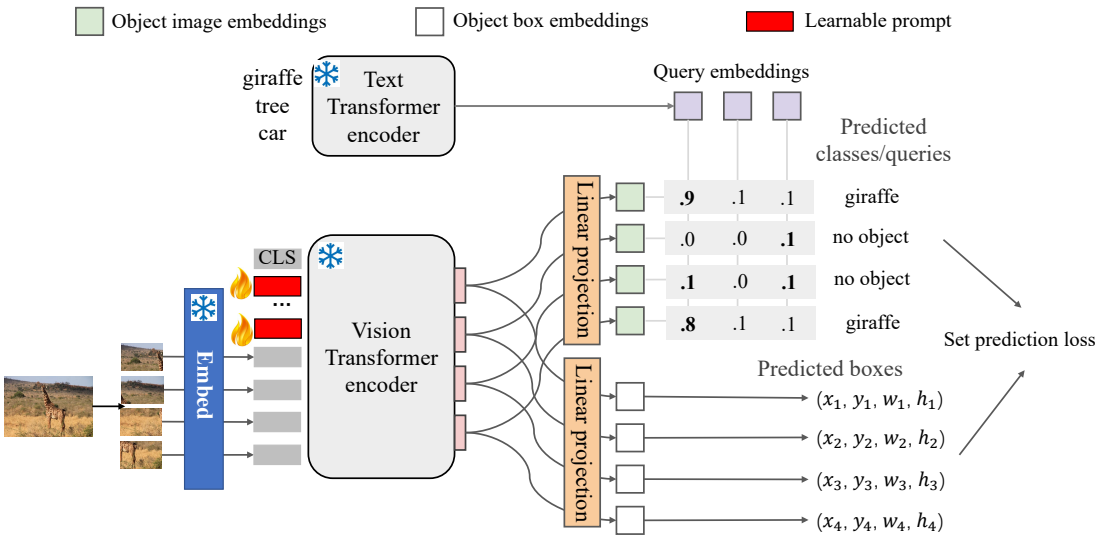

Figure 18. The architecture of OWL-ViT.

prove the precision of query data. The architecture of CLIP is shown in Fig. 20.

8.4.2 Training

We update the CLIP through the deep soft prompt tuning [18]. 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 as-

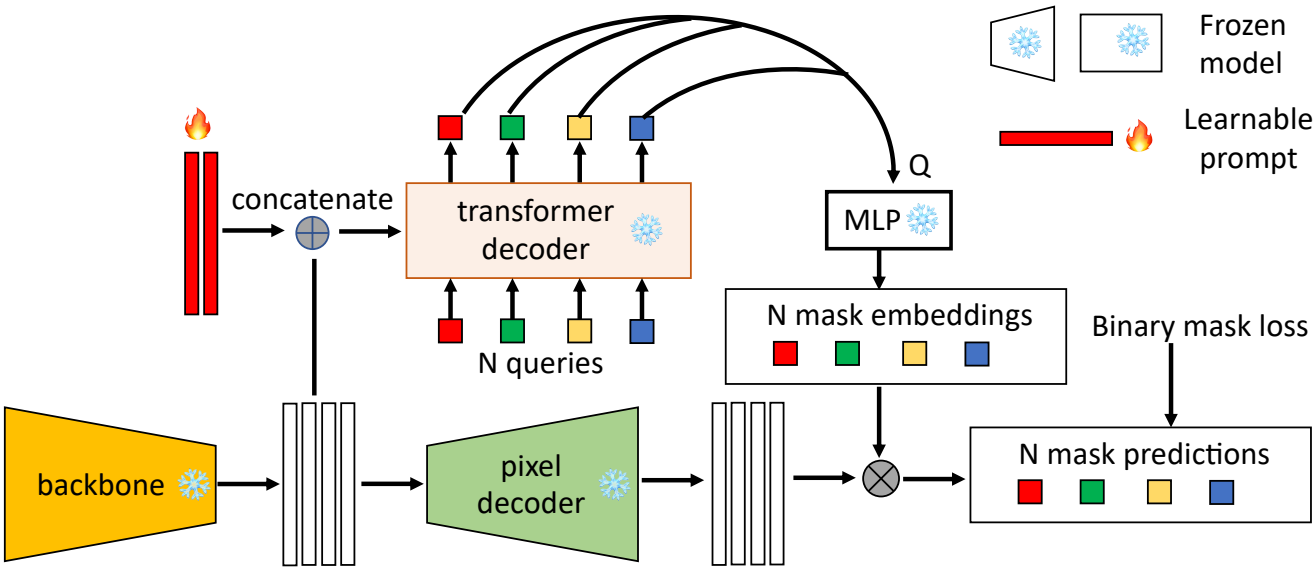

Figure 19. The architecture of Maskformer.

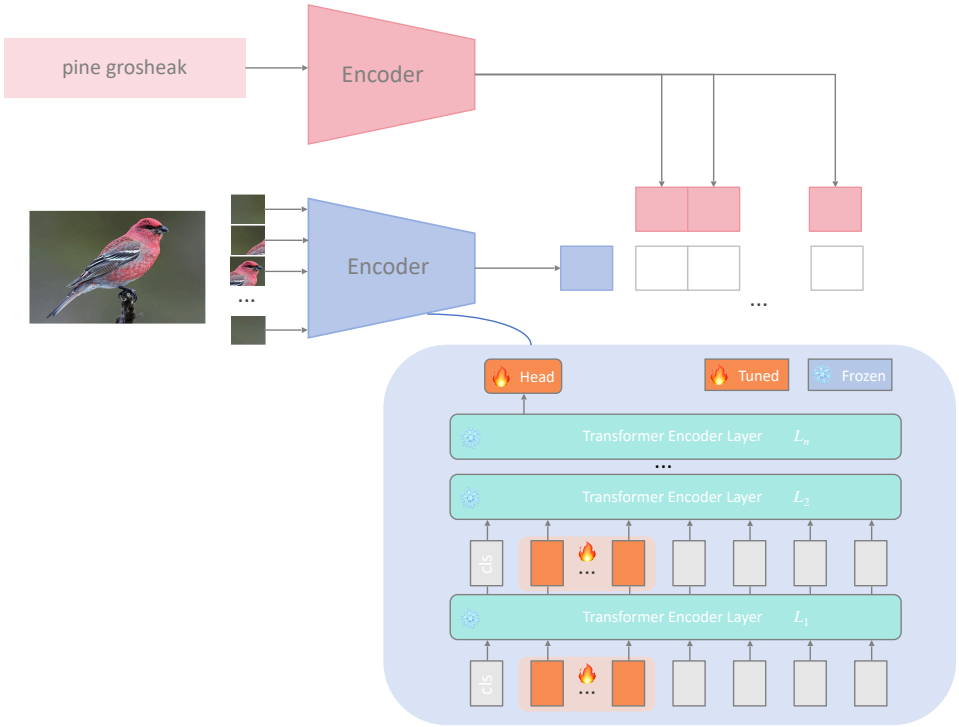

Figure 20. The architecture of CLIP.

sociated 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 soft prompt learning. We conduct prompt training 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.

8.4.3 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 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.

8.5. Update REPLACE tool

We use Stable Diffusion [48] 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 vector for training images and store it. The dimensionality of the prompt vector is 768. The architecture of Stable Diffusion is shown in Fig. 21.

8.5.1 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 [48] is minimized. Subsequently, the 768-dimensional prompt obtained from the training stage is stored in the prompt pool. In terms of experimental setup, we employ the Adam optimizer. Through our experimentation, we find that setting the learning rate to $5e-3$ achieves the best learning results.

8.5.2 Inference

Similar to other visual models, 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.

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 models	50.2	53.0	57.8
NLVRv2	Baseline	50.0	60.2	64.8
	+ Update LLMs	57.4	61.0	66.4
	+ Update visual models	61.6	62.6	67.4

Table 7. Different LLMs on the online learning setting using the GQA and NLVRv2 datasets.

9. More Experimental Results

9.1. 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. 22. 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.

9.2. 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 GQA and NLVRv2 datasets are used. Results are shown in Tab. 7. Similar to the offline setting in Section 4.2, updating LLMs and VFMs both leads to improvements.

9.3. 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. 23. The reflection and learning phases for the SELECT tool are shown in Fig. 24. The reflection and learning phases for the LOC tool are shown in Fig. 25. The reflection and learning phases for the REPLACE tool are shown in Fig. 26. The reflection and learning phases for the CLASSIFY tool are shown in Fig. 27. The reflection and learning phases for the SEG tool are shown in Fig. 28.

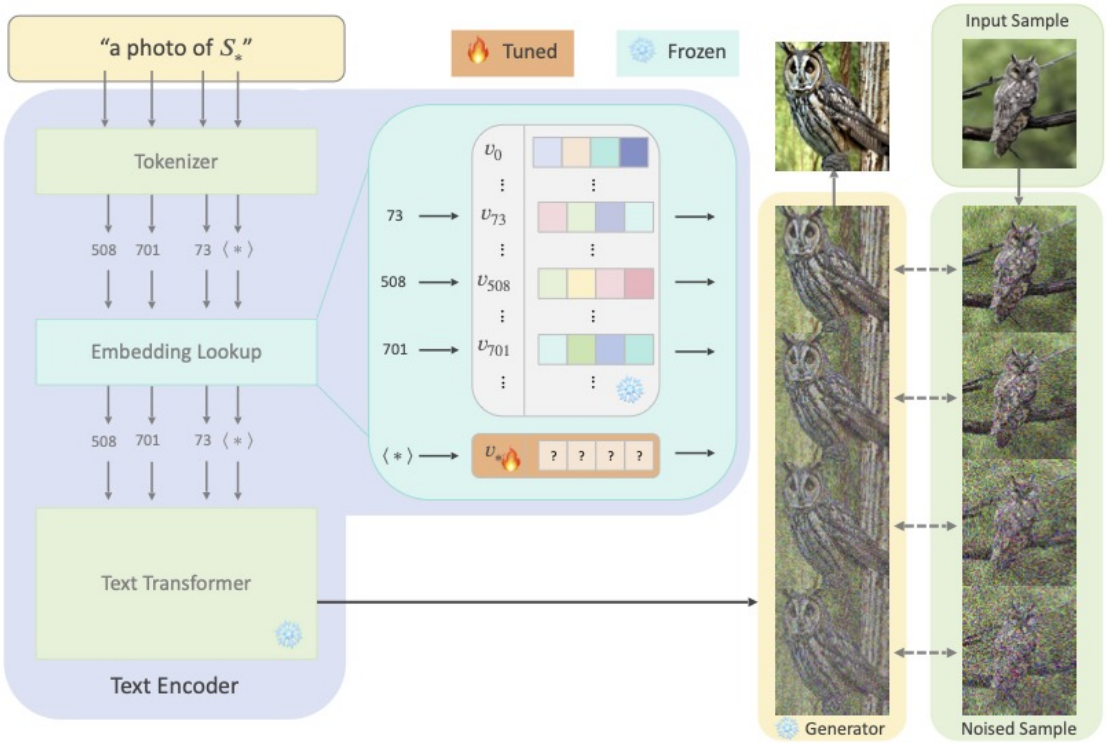

Figure 21. The architecture of Stable Diffusion.

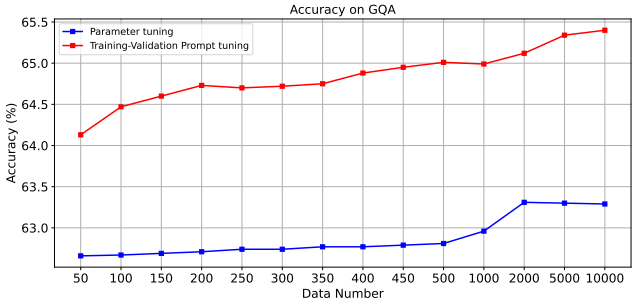

Figure 22. Accuracy curves on the GQA dataset

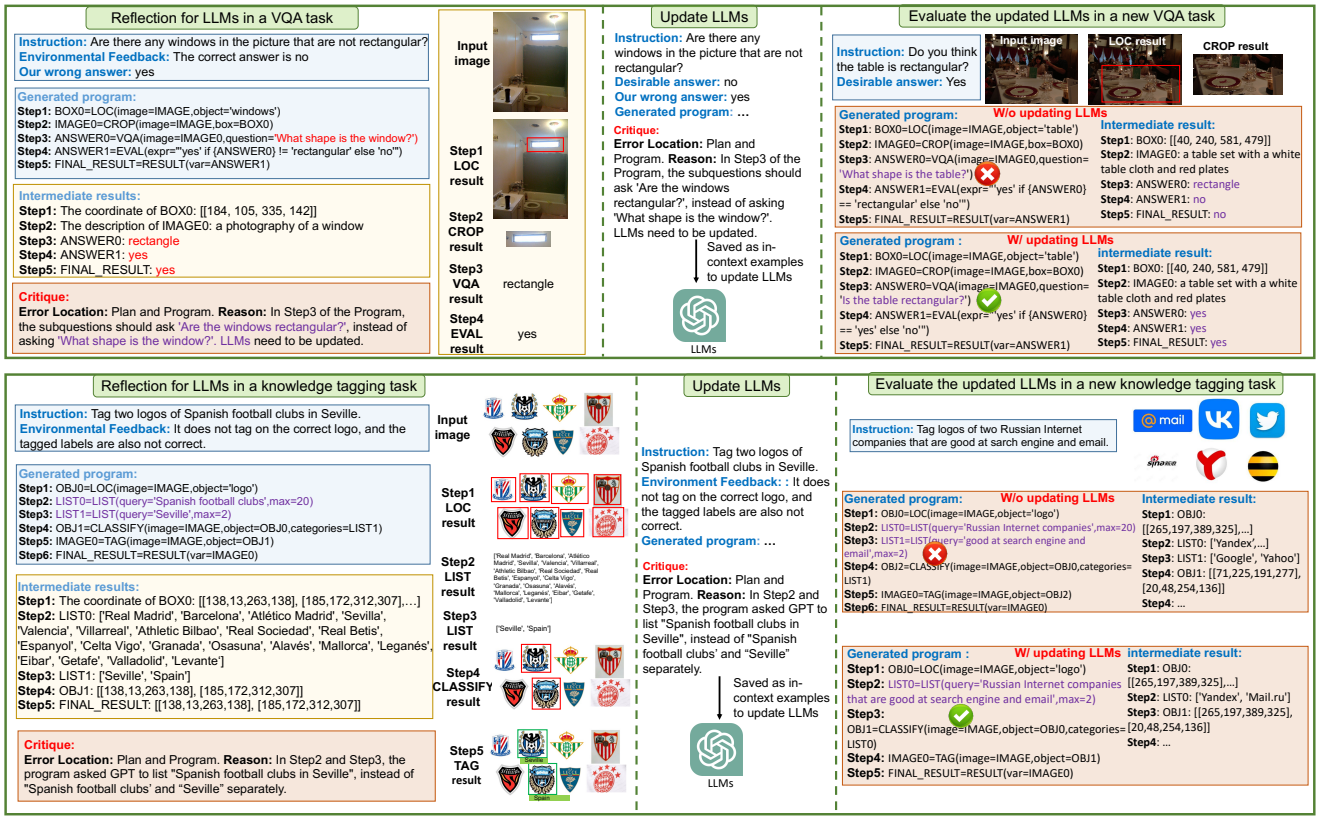

Figure 23. Case studies of updating LLMs.

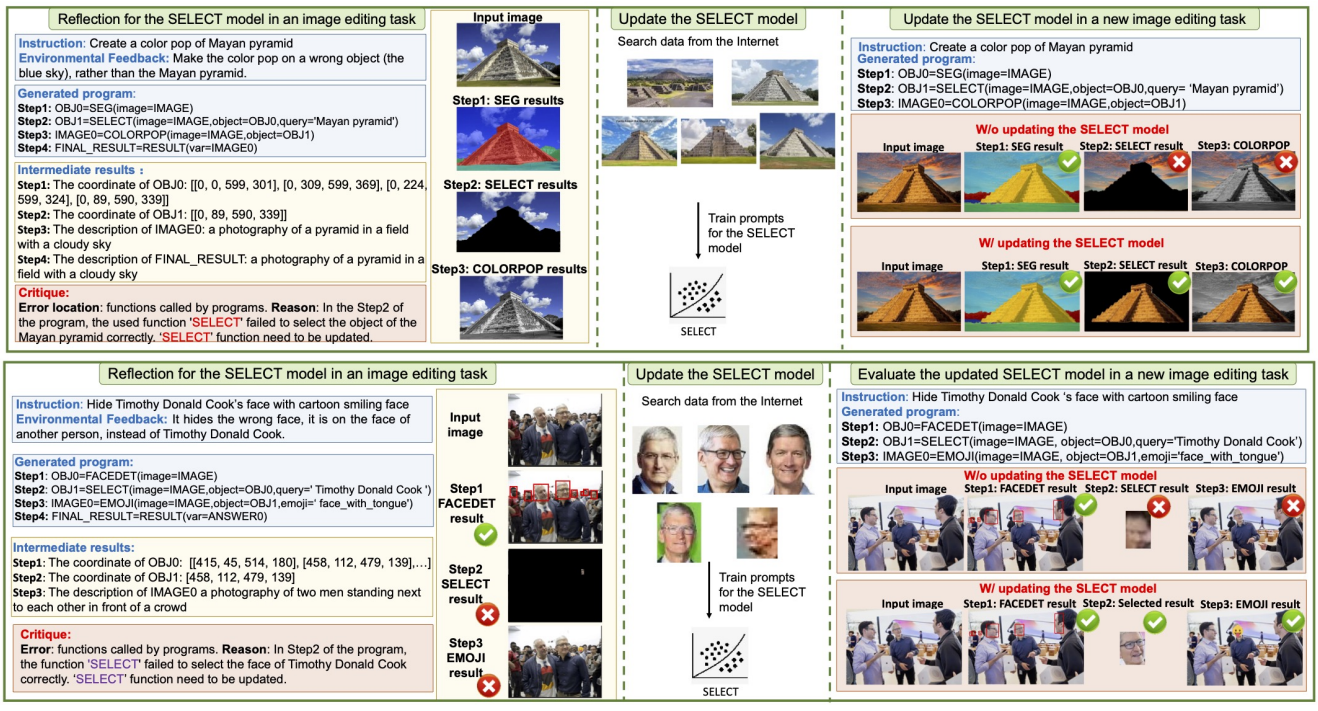

Figure 24. Case studies of updating the SELECT tool.

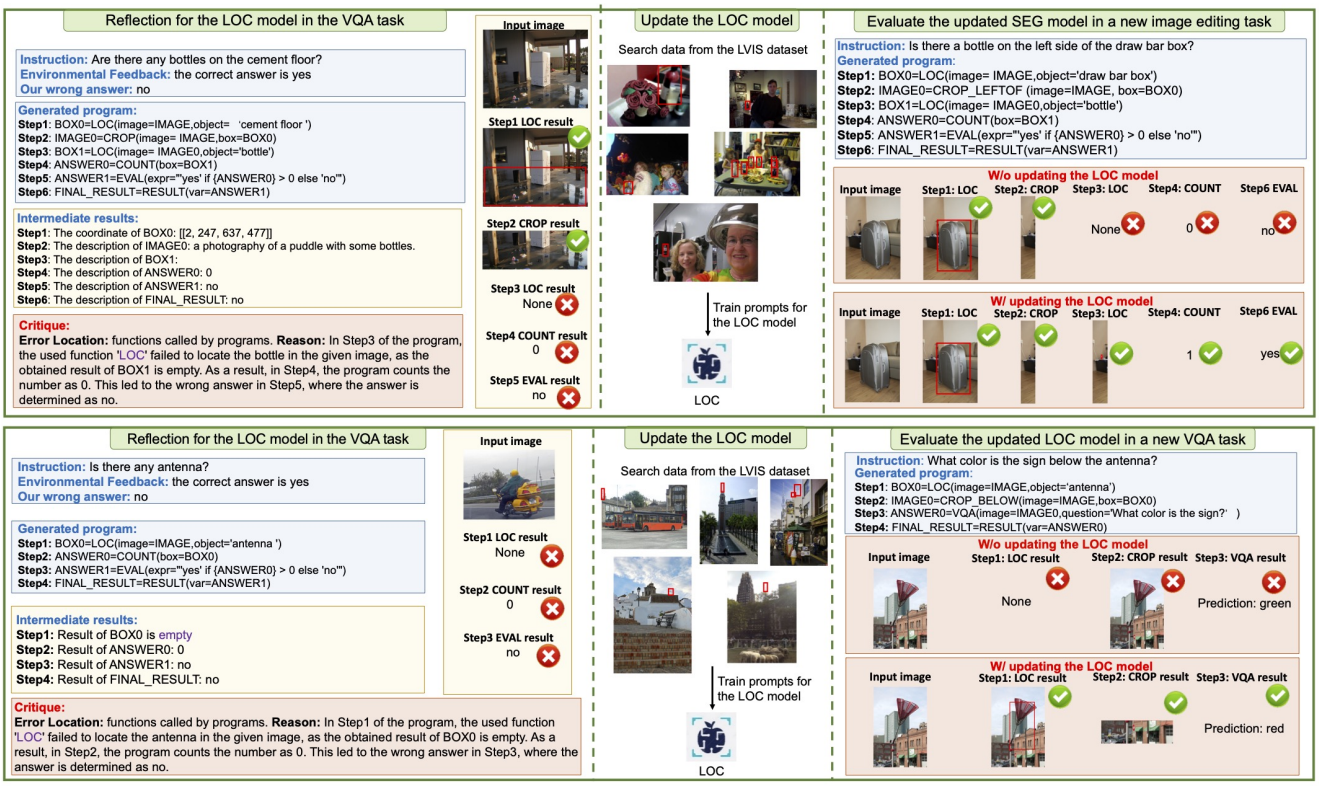

Figure 25. Case studies of updating the LOC tool.

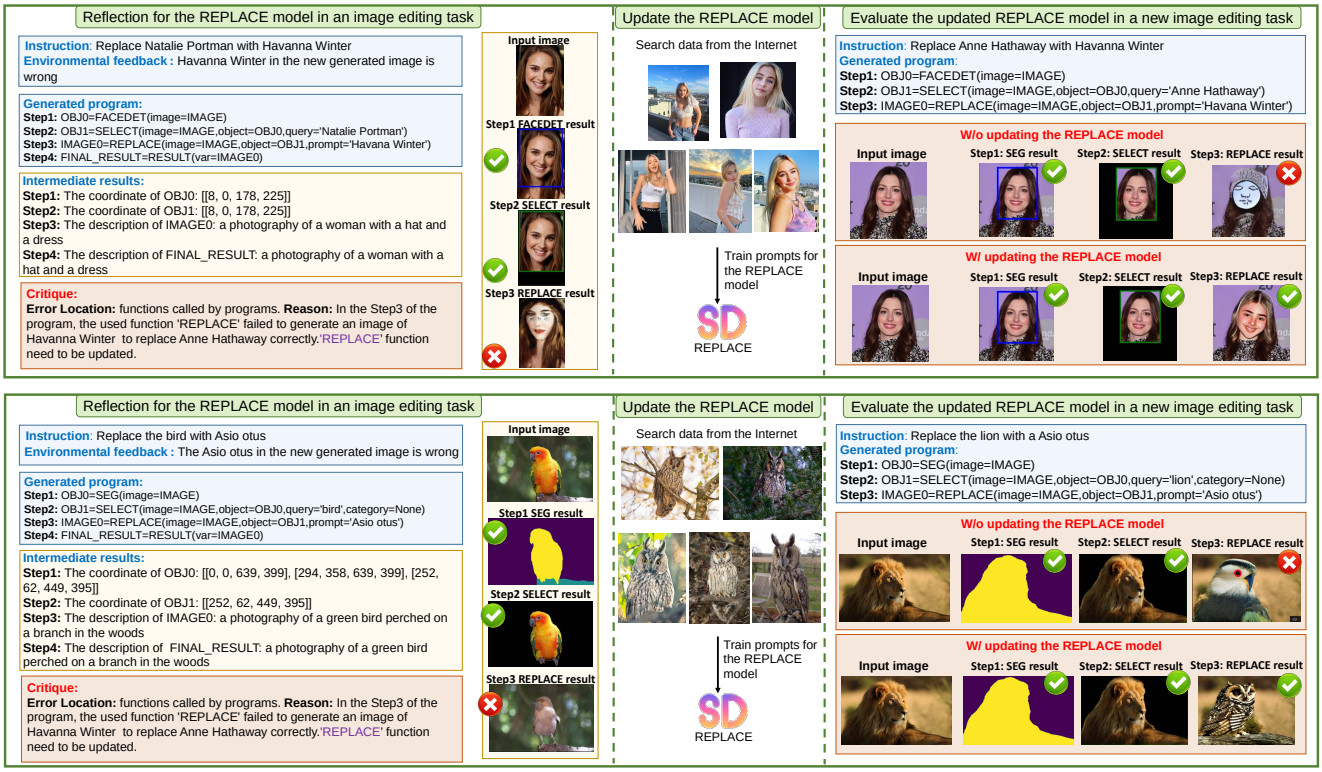

Figure 26. Case studies of updating the REPLACE tool.

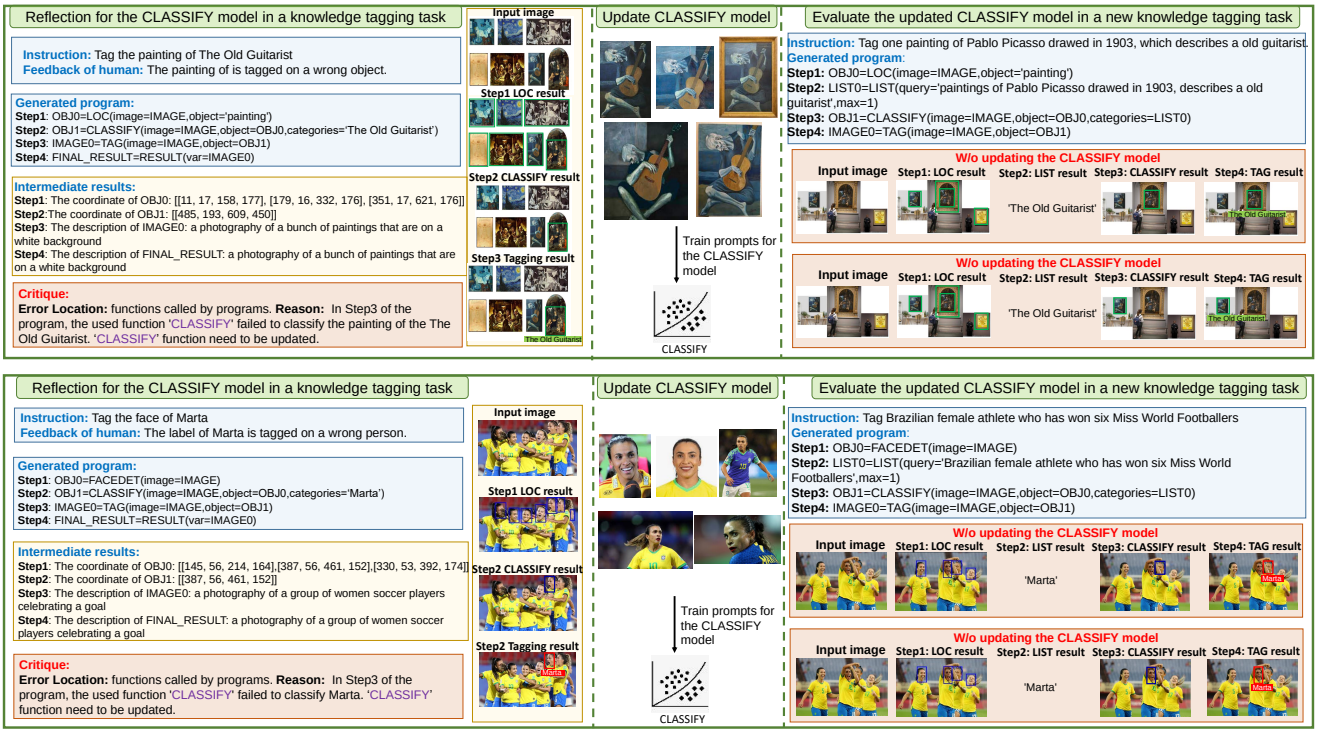

Figure 27. Case studies of updating the CLASSIFY tool.

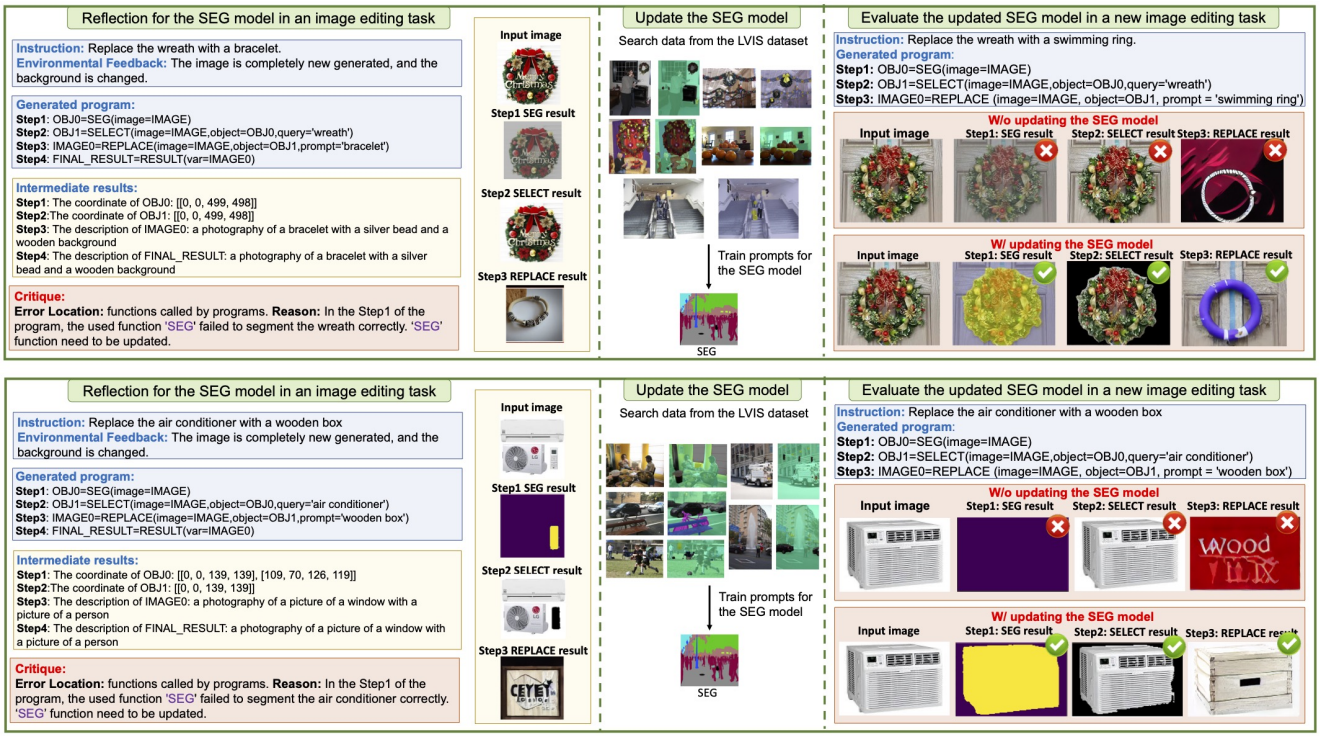

Figure 28. Case studies of updating the SEG tool.