HYDRA: A Hyper Agent for Dynamic Compositional Visual Reasoning

Supplementary Material

We provide additional details about our framework, HYDRA, as supplementary material. We provide more details about the number of instruction samples, our approach to training RL agents, the templates used in the textualizer module, additional qualitative analysis examples, and prompts employed in the LLMs, in further detail below.

A Number of Instruction Samples

As outlined in approach section 3, the LLM in our planner generated a set of N instruction samples. We conducted an experiment using the OK-VQA dataset to determine the optimal number, which is detailed in Figure 6. In our experiment, we tested various sample sizes, namely N = 1, 3, 5, 7, and 10. Upon examining the results presented in Figure 6, we observed that employing 10 samples posed challenges for the RL agent's training and alignment with the complete set of actions. This challenge is particularly pronounced due to the modest size of the neural network composing the RL agent, essentially consisting of an MLP. As the number of instruction samples grows, and taking into account the limited dataset and the modest scale of the RL agent, incorporating a larger quantity of instruction samples becomes increasingly difficult for the small RL agent, consequently leading to a decline in performance. Conversely, with a very low number of actions (e.g., 1), the agent can request more instruction samples if it finds the current ones invalid, eventually obtaining a good sample. While a small number of actions makes it easier for a small RL agent to converge on meaningful decisions, it also increases the likelihood of rejecting all instruction samples compared to having five samples. Therefore, in terms of performance

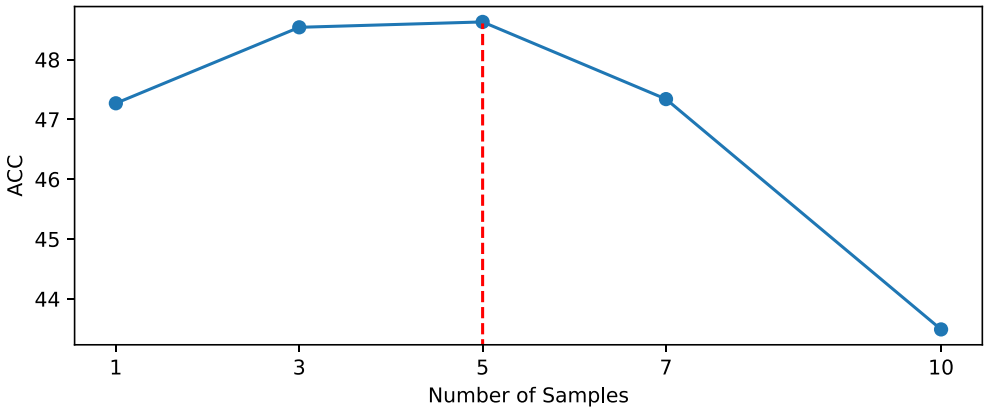

Fig. 6: Experimenting with different instruction sample sizes on the OK-VQA dataset. The vertical axis represents accuracy, while the horizontal axis denotes the number of instruction samples.

and efficiency, using five samples appears more promising, as depicted in the figure, which yielded an accuracy of 48.63%.

B RL Agent Training

As described in the approach section 3, the controller module comprises an RL agent implemented as an MLP utilizing the DQN algorithm [27]. Below, we offer a more detailed explanation of the RL agent training process and its interaction with the environment, State Memory Bank and Meta Information, throughout.

B.1 Embedding

In the training phase, RL agent needs to interact with an environment, State Memory Bank and Meta Information, which contains the current state and comprehensive information from the previous iteration. Meta information may encompass the system's own skills, its functionalities, and a description of a task (e.g. query). In HYDRA, we utilize a text-to-embedding template along with embedding models from the OpenAI API to obtain environmental information. This approach enables the learning of effective control policies from textual data within complex RL environments tailored for visual reasoning tasks.

B.2 Training process

We will detail the training process of the RL agent in this part. Initially, we configure a multi-layer perceptron with dimensions {1536, 512 and $N+1$} with the use of *text-embedding-3-small* model from the OpenAI API (N is the number of instruction samples, mentioned in section 3). The learning start threshold is set to 1000, indicating that the controller will make decisions randomly during the first 1000 observation processes. We denote observation counts as ω. The exploration epsilon is set at 0.2, with an exploration epsilon decay rate of 0.02, and the epsilon decay interval is established at 200 steps. Therefore, the exploration threshold value is as:

$$\mathcal{T} = \frac{0.2}{0.02 \times \frac{\omega}{200}} \tag{5}$$

To enhance learning in visual tasks, the controller occasionally opts for random exploration over following the learned policy when a randomly generated value falls below the exploration threshold $\mathcal{T}$ within the range $[0, 1]$. As explained in the training phase in our approach, rewards for each action are calculated across a variety of environmental scenarios, with all corresponding rewards and situations stored in a reward buffer. We have optimized the batch size to 128, and the learning rate is established at 1×10^{-4}. During each weight update cycle, a batch of samples, matching the batch size, is randomly drawn from the reward buffer to update the MLP's weights. For further details on the DQN updating process employing stochastic gradient descent and reward buffer, please refer to [27].

C Ablation Study

ViperGPT. In this experiment, we follow the original ViperGPT official GitHub repository for all datasets. Note, however, that ViperGPT uses Codex, which is deprecated. Therefore, in our experiment, we replace Codex with GPT-3.5 Turbo-0613.
HYDRA-IR-S. The RL agent has been integrated to ViperGPT, providing it with the ability to make decision on keeping or re-generating the instruction from LLM. This integration aims to enhance the model's decision-making capabilities by allowing it to learn optimal policies through trial and error. With this addition, the ViperGPT achieved 5.77% improvement on its results as shown in Table 5 (row 2).
HYDRA-RL-IR. In this experiment, unlike the ViperGPT model, we asked the LLM to generate more than one instruction sample. As shown in Table 5 (row 3), the performance increased by 2.5% in terms of accuracy, reaching 39.84%.
HYDRA-IR. In this experiment, we removed Incremental Reasoning, which means the model no longer processes information incrementally or adaptively over multiple steps. This removal likely impacts the model's ability to reason and solve complex tasks that require multi-step reasoning or context-dependent decision-making. Consequently, the accuracy decreased slightly to 45.98% as shown in Table 5 (row 4).
HYDRA-RL-S. In this experiment, we removed sampling, meaning the model's LLM only generates one instruction sample, and the RL agent has been eliminated from our framework. As shown in Table 5 (row 5), the model benefited from this adjustment, achieving an accuracy of 41.08%.
HYDRA-S. In this experiment, we removed sampling, meaning the model's LLM only generates one instruction sample. As shown in Table 5 (row 6), the removal of sampling negatively impacts the model's performance.
HYDRA-RL. Similar to the previous experiment, the RL agent has been eliminated from our framework. This removal removes the model's ability to learn from rewards and adjust its behavior accordingly, potentially limiting its capability to perform tasks that require adaptive decision-making or exploration of the environment. Despite this, the model still achieved an accuracy of 46.93% as shown in Table 5 row 7.

D Fail Rate Analysis

To analyse model stability, following the protocol suggested by [9] we manually reviewed $\sim$ 100 samples per dataset and categorized error sources into four groups as shown in Fig. 7. When the LLM can not provided any valid instruction, HYDRA's performance suffers as the controller can not select good instructions. This error type is the most common in GQA. Moreover, code generator problems like calling non-existent APIs can also impact stability, as seen in RefCOCO datasets. Therefore, using a more powerful LLM, e.g. GPT-4, can mitigate the impact of planner constraints and code generator issues and improve HYDRA's

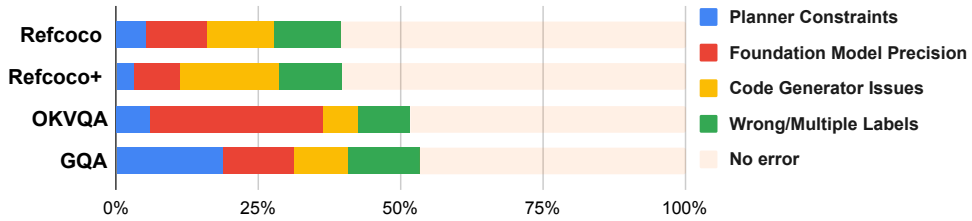

Fig. 7: HYDRA Fail rate on each dataset.

Table 6: HYDRA Performance on GQA Dataset

Model	GQA ACC(%)	A-OKVQA ACC(%)
BLIP2 [19]	45.5	53.7
HYDRA w BLIP2	**47.9**	**56.4**
LLaVA1.5 (7B) [22]	62.0	61.6
HYDRA w LLaVA1.5 (7B)	**64.5**	**62.5**

performance. Additionally, insufficient precision of the foundation models also leads to errors, as shown in OKVQA, indicating the need to employ SoTA foundation models.

E Plug and Play in HYDRA

HYDRA enhances visual reasoning tasks by leveraging its inherent capability to employ any foundation model as a VFM API. Intuitively, HYDRA's performance can be further improved by integrating the recent and larger foundation models, thus surpassing the performance of using a sole foundation model for the same task as shown in Table 6. This is because HYDRA has the ability to determine the appropriate API and utilize it at the correct step in the reasoning process. The experiment result Shows an improved accuracy of 62.5% on the A-OKVQA dataset and 64.5% on the GQA dataset when employing LLaVA-1.5. These results underscore the benefits of integrating cognitive agents for enhanced visual reasoning tasks and emphasize the advantages of the autonomy mechanism inherent in the compositional approach.

F Textualizer Module Templates

In the approach Section 3, it is mentioned that when the perceptual output from the reasoner module is incomplete or unsuccessful, it undergoes conversion to textual format within textualizer module, as depicted in Figure 2. The perceptual output from the reasoner, which may consist of bounding boxes, verifications, or captions, is transformed into a textual format using some templates. These templates are provided in Template C which demonstrates the conversion

of visually grounded fine-grained information into textual format. For instance, when the perception function *find* is activated, the name of the target object is recorded in the detection results and the number of detected target objects is documented. Moreover, the bounding box coordinates for each target object are also recorded. The number of target objects and their locations, such as bounding box coordinates, provide crucial information to the planner and controller for their subsequent actions. Similarly, upon activating the perception function *exists*, the system records the name of the object being checked, the name of the image, and the outcome of the check.

Template C: Feedback Summarizer Examples

find
Detection result: Only one {object_name} has been detected in {image_name}.
Detection result: {num} {object_name} have been detected in {image_name}.
Detection result: no {object_name} has been detected.
Detected bounding box [x1,y1,x2,y2]: {object_name}_{current_img_no} in {image_name} is {bd_box_prediction};
existence
The existence of {object_name} in image patch {image_name} is: {exist_result}.
verify
The verification of {category} in {image_name} is: {verification_result}
caption
The caption for image patch {image_name} is: {caption}.
simple question answer
The answer for image patch {image_name} in response to the question '{question}' is: {query_answer}
depth calculation
The median depth for image patch {image_name} is: {median_depth}
LLM answer
The obtained answer from LLM to the question, {query} with the additional context of {context} is: {return_answer}
sort
The patches list has been sorted from left to right (horizontal). Now, the first patch in the list corresponds to the leftest position, while the last one corresponds to the rightest position.
get middle patch
The {name} is the middle one in the list.
get the closest patch
The {name} is the closest one to {anchor_name}.
get the farthest patch
The {name} is the farthest one to {anchor_name}.
variables
{variable_name}: {variable_value}

Fig. 8: More qualitative result examples from HYDRA.

G More Qualitative Analysis Examples

In this section, we offer a more qualitative analysis showcasing the output of each step in HYDRA, as illustrated in Figure 8. As depicted, the input image and query in the blue box are presented. Yellow boxes display the instructions step by step, while the green one shows their corresponding intermediate results.

H LLM's Prompts

In HYDRA, we utilized LLMs in three distinct modes, as outlined in Section 3: as an instruction sample generator in the planner, a code generator in the reasoner, and for summarizer in the textualizer module. For each mode, the prompt used is defined herein, and we provide detailed information on each prompt. Prompts H.1, H.2, and H.3 illustrate the abstract format of corresponding prompts for the instruction sample generator, code generator, and summarizer, respectively.

- Prompt H.1: An instruction prompt is a crucial tool that informs the planner about the available perception skills, $\pi \in \Pi$ demonstrates the utilization of these skills, and describes how an instruction can leverage various skills for effective execution. Within the meta information, the skills are directly conveyed to the planner, informing it of HYDRA's capabilities and guiding it towards generating appropriate subsequent instructions. Otherwise, the planner might generate instructions that cannot be executed.
- Prompt H.2: A code prompt serves as a vital instrument, briefing the reasoner on the available perception skills along with templates for Python classes and functions. It includes a Python class and outlines several functions, showcasing the methodology for formulating Python code in response to the received instructions.
- Prompt H.3: A prompt functions as a guiding template, enabling the LLM to assess the adequacy of fine-grained information provided, based on the current state stored in State Memory Bank. If the detailed grounding information is adequate to address the query, the LLM will directly produce the answer, thus eliminating the need for further sequential responses.

Prompt H.1: Instruction Generation

[META_INFO]
How to Use these Skills ideally: [EXAMPLE]
Now the demonstration has ended. The following information are provided to you for recommending next-step instructions.
About Query: [QUERY_TYPE]
Current Step: [CURRENT_STEP_NUM]
All Previously Taken Instruction:
[INSTRUCTION_HISTORY]
Executed Python Code:
image_patch = ImagePatch(image)
[CODE_HISTORY]
Each variable details: [VARIABLE_AND_DETAILS]
Execution Feedback (Details of the known visual information in the image): [FEEDBACK_HISTORY]
The question is '[QUERY]'
Please, provide [NUMBER_OF_SAMPLES] alternative instructions and associate each with a probability value indicating its likelihood of leading to the final answer.

If available information is sufficient for answering question, please directly provide final answer as response.
Your response is here:

Prompt H.2: Code Generation

[META_INFO]
Provided Python Functions/Class:
[PYTHON_API_CODE]
Please only return valid python code: If a Python variable is not found in the 'Executed Python Code' section, it means that variable does not exist, and you cannot use any variable that has not been defined in the 'Executed Python Code'.
[EXAMPLE]
Now the demonstration has ended. An instance (image_patch = ImagePatch(image)) of the ImagePatch class is provided.
Please translate only the 'Current Instruction' into Python code. If the 'Current Instruction' mentions the final process, assign the result to the variable named final_answer for the concluding statement. If there is no mention of a final process in the current instruction, refrain from using final_answer. If a Python variable is not found in the 'Executed Python Code' section, it means that variable does not exist, and you cannot use any variable that has not been defined in the 'Executed Python Code'. About Query: [QUERY_TYPE]
Query: [QUERY]
Current Step: [CURRENT_STEP_NUM]
All Previously Taken Instruction:
[INSTRUCTION_HISTORY]
Executed Python Code: image_patch = ImagePatch(image)
[CODE_HISTORY]
Each variable details:[VARIABLE_AND_DETAILS]
Execution Feedback (Details of the known visual information in the image): [FEEDBACK_HISTORY]
Current Instruction: [CURRENT_INSTRUCTION]
Generated Python Code for Current Instruction [CURRENT_INSTRUCTION] here:

Prompt H.3: Summarizer

[META_INFO]
Below is the information related to the question along with known visual details
About Question: [QUERY_TYPE]
All Previously Taken Instruction:
[INSTRUCTION_HISTORY]
Executed Python Code:
image_patch = ImagePatch(image) [CODE_HISTORY]
Each variable details: [VARIABLE_AND_DETAILS]
Execution Feedback (Details of the known visual information in the image): [FEEDBACK_HISTORY]
The question is '[QUERY]'

You need to base on details of the known visual information in the image to answer question. Respond concisely with key terms or names related to the question. Base your deductions solely on the Execution Feedback, which provides details of the known visual information in the image. Avoid making any random guesses if the available evidence does not sufficiently support your answer. For example, When provided with limited information that only identifies an object as a fruit without further details, it's crucial to avoid making arbitrary guesses about the fruit's identity. Instead, the response should acknowledge the insufficiency of the data for a definitive identification. If available information is insufficient for a definitive answer, reply with 'continue'.
Your answer is here: