

DWIM: Towards Tool-aware Visual Reasoning via Discrepancy-aware Workflow Generation & Instruct-Masking Tuning

Supplementary Material

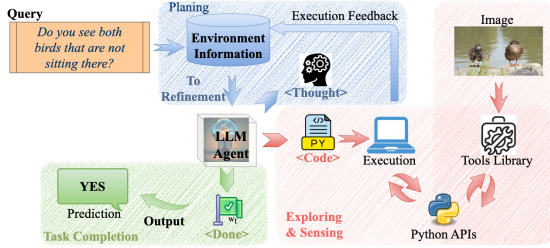


Figure 5. Auto-Exploring Agentic Framework. The LLM agent generates `<Code>` for execution, `<Thought>` for reasoning, or `<Done>` to complete the task. It dynamically generates or refines actions while storing environmental information for incremental reasoning.

6. Auto-Exploring Agentic Framework

Our framework dynamically generates `<Code>`, `<Thought>`, or `<Done>` without a fixed pattern (e.g., Code follows Thought in CodeAct [58], or Act follows Thought in ReAct [65]). The LLM generates `<Code>` for all execution steps involving tool usage. If reasoning is required or an ineffective action is detected by the discrepancy-aware recognition step, the model outputs the corresponding information in `<Thought>`. This flexibility makes our model inherently dynamic.

7. DWIM Qualitative Analysis

In this section, we provide a qualitative analysis showcasing the output of each step in DWIM, as illustrated in Figure 6. The input image, located at the top-left of each bounding box, and the query are displayed in the light blue box. The purple box displays the `<Thought>`-action, the yellow box shows the `<Code>`-action, and the pink box presents the environment feedback, for each turn respectively. In many cases, LLaVa-1.5 [37], one of the tools in our tool library, fails to answer the question. In comparison, by leveraging the tool-awareness ability, DWIM provides correct answers by utilizing tools better suited for the question.

8. “Standard” v.s. “Discrepancy-aware” Training Workflow Generation

In this section, we provide a qualitative analysis of the differences between standard training workflow generation and discrepancy-aware training workflow generation, as shown in Figure 7. Using the standard method, the model assumes environmental feedback is always correct under the same auto-exploring framework. As a result, the standard



Figure 6. DWIM Qualitative Result Example

method does not check for discrepancies between feedback information and the answer, leading to failed workflows due to tool errors and preventing the generation of a viable workflow for that training data point. Consequently, a large portion of training data lacks correct workflows that yield the right answers and is discarded, resulting in high data waste. In contrast, discrepancy-aware training workflow generation accounts for discrepancies between each feedback step and the answer to ensure that actions remain valid while continuously refining workflows to reach the correct final answer. This discrepancy-aware step also enables the model to recognize when tools provide incorrect information, which is crucial for training agent tool awareness.

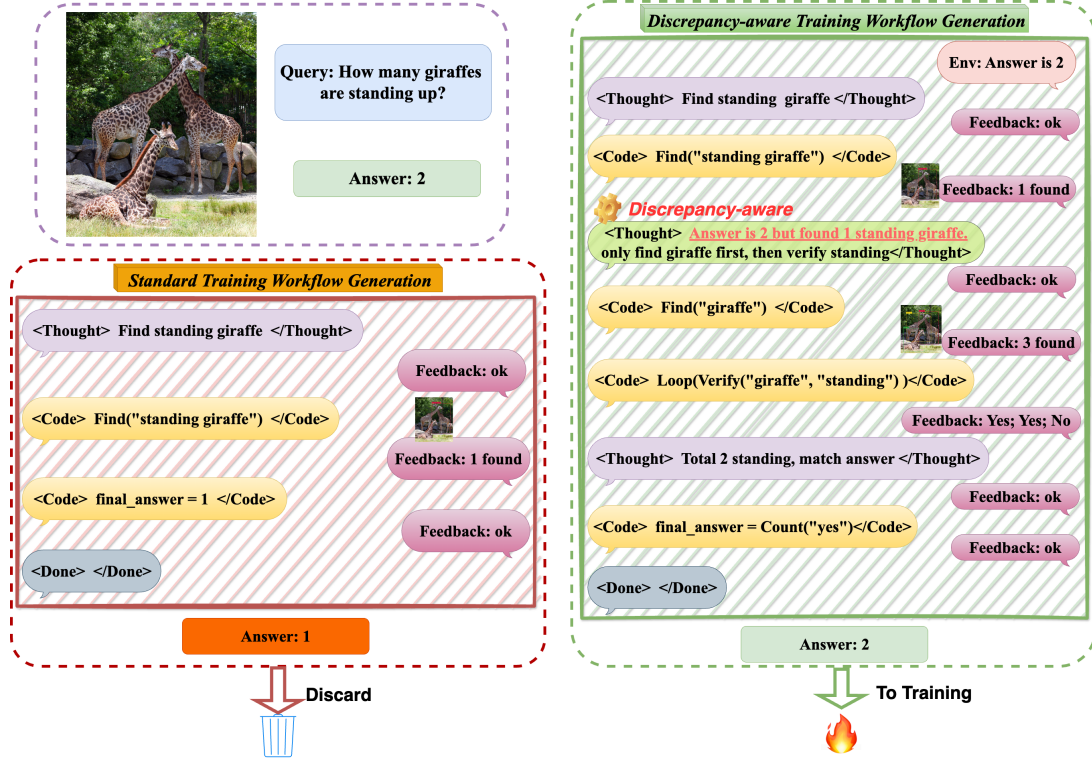


Figure 7. “Standard” v.s. “Discrepancy-aware training” Training Workflow Generation

9. Analysis of Action Flagging

In DWIM, flagging action effectiveness based on both LLM assessments and environment feedback is a prerequisite for instruct-masking. The LLM identifies discrepancies between feedback and the expected answer by generating descriptive sentences (e.g., ω_{Rethink}). We assess action effectiveness using both the content of these sentences and the corresponding feedback. To support this, we employ a rule-based method that flags ineffective actions based on discrepancy-aware recognition and environmental feedback. These flagged actions are excluded from masking, preventing the model from learning from mistakes. However, LLM assessment output may not fully adhere to the output template when recognizing ineffective actions in a workflow due to its complexity, which involves natural language, code, and intricate environment feedback, potentially leading to misflagging.

To evaluate our proposed flagging method, we conduct a human evaluation to assess the effectiveness of flagging in 100 workflow samples generated using the discrepancy-aware training workflow generation method from the GQA training set, comparing the results with our rule-based approach. In these 100 workflow samples, 52.1% are

$\langle \text{Code} \rangle$ -actions, 23.5% are $\langle \text{Thought} \rangle$ -actions, and 24.4% are $\langle \text{Done} \rangle$ -actions.

In DWIM, any $\langle \text{Code} \rangle$ -action flagged by environment feedback as “Traceback” is flagged as ineffective. Similarly, a action preceding a $\langle \text{Thought} \rangle$ -action (e.g., ω_{Rethink}) with the context “however” or “rethink” is also considered ineffective. An action that is logically correct but produces an incorrect result will trigger a discrepancy-aware $\langle \text{Thought} \rangle$ -action. A total of 41 ineffective $\langle \text{Code} \rangle$ -actions, 3 actions preceding a $\langle \text{Thought} \rangle$ -action with the context “rethink,” and 26 discrepancy-aware $\langle \text{Thought} \rangle$ -action were flagged as ineffective. In the human evaluation, we used the majority vote from three evaluators and obtained the same results for normal ineffective $\langle \text{Code} \rangle$ -actions. Additionally, 3 more actions triggering “rethink” or “replan” $\langle \text{Thought} \rangle$ -actions and 2 additional discrepancy-aware $\langle \text{Thought} \rangle$ actions were identified.

As illustrated in Figure 8, all normal ineffective $\langle \text{Code} \rangle$ -actions were flagged; however, only 50% of ineffective $\langle \text{Thought} \rangle$ -action preceding a $\langle \text{Thought} \rangle$ -action with the context “rethink” were detected. Although such ineffective actions constitute only around 10% of the total sample actions, it is crucial that they are not masked and are properly learned. The current rule-based flagging method is not en-

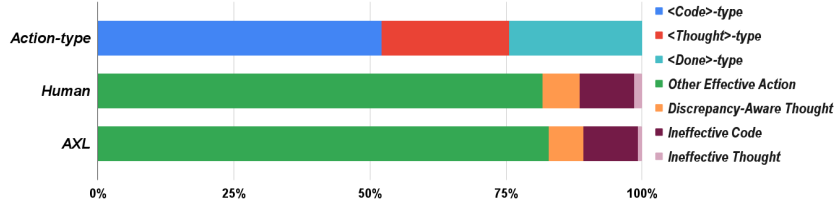


Figure 8. DWIM and Human Flagging of Ineffective Actions on Collected Workflows. DWIM’s flagging results are close to those of human evaluators; however, there is still room for improvement, particularly in flagging actions that trigger “rethink.”

Table 8. Tools’ Functionality

Tools	Model Name				Description
	LLaVa-1.5-7B	BLIP2-Flan-T5-XXL	GPT-4o-2024-05-13	GroundingDINO-Base	
Detector				✓	Detect Object
Check Existence				✓	Check Object Existence
Simple Query Answer	✓	✓			Answering Simple Questions with a Word or Phrase
Complex Query Answer	✓				Answering Complex Questions with a Sentence
Captioning	✓				Get Image Caption
Acquiring External Knowledge			✓		Acquire External Knowledge
Boolean to Yes/No					Convert True/False to Yes/No
Image Crop					Crop Images Based on Provided Coordinates
Property Matching	✓				Identify the Best-Matching Visual Property
Verify Property	✓				Verify Visual Property

Table 9. Task-specific Tool Library.

Tools	Tasks					
	VCR	EKVQA	VLCU	VASA	GD	CCQ
Detector	✓	✓	✓	✓	✓	✓
Check Existence	✓	✓	✓	✓	✓	✓
Simple Query Answer	✓	✓	✓	✓	✓	✓
Complex Query Answer		✓		✓		
Captioning	✓	✓	✓	✓	✓	✓
Acquiring External knowledge		✓				
Boolean to Yes/No	✓		✓	✓		
Image Crop	✓	✓	✓	✓	✓	✓
Property Matching	✓				✓	
Verify Property	✓	✓	✓	✓	✓	✓

tirely precise, particularly in complex contexts. In future work, we aim to develop an LLM-based flagger for more accurate flagging of ineffective actions by leveraging environment feedback and recognition results.

10. Additional Ablation Study

We conducted an experiment where the answer is given but discrepancies are not recognized (annotated as *Given Y*) as shown in Table 10. While *Given Y* produces more workflows than the standard method, it does not outperform our approach. Moreover, a portion of its successful workflows result from directly copying the answer.

11. Additional Tool Awareness analysis

We evaluate models’ tool awareness based on overall performance and tool utilization efficiency, as described in Section 4. To further evaluate the improvement in tool aware-

Table 10. Additional Ablation Study: Effect of *Given Y* During Workflow Generation on GQA

Fine-tune	Training Workflow Generation	Data Utilization (%)	GQA (%)
SFT	Standard	48.2	53.6
Instruct-Masking	Standard	48.2	57.9
SFT	Given Y	60.3	54.3
Instruct-Masking	Given Y	60.3	60.9
SFT	Discrepancy-aware	68.3	54.8
Random-Masking	Discrepancy-aware	68.3	65.1
Masking-W-Rethink	Discrepancy-aware	68.3	68.0
Instruct-Masking	Discrepancy-aware	68.3	69.3

ness of DWIM compared to a frozen LLM, we conduct a human evaluation on 100 workflow samples from GQA evaluation results for each model. Specifically, we examine the proportion of generated workflows that should yield correct answers if the tools function accurately but fail in practice, as well as the proportion of workflows that are logically incorrect.

The evaluation results indicate that 71% of DWIM-generated workflows and 47% of frozen LLM-generated workflows produced correct answers. Additionally, 18% and 28% of workflows, respectively, should yield correct answers but failed due to tool errors. Furthermore, 7% of DWIM-generated workflows and 23% of frozen LLM-generated workflows were logically incorrect. Lastly, 4% and 2% of workflows, respectively, produced correct answers but were misclassified as incorrect due to evaluation metric errors.

Based on our investigation, we observe a significant improvement in overall performance after training, indicating

the effectiveness of the generated workflows. Additionally, the average tool utilization per query decreases, suggesting improved efficiency. Moreover, DWIM has a 10% lower failure rate than the frozen LLM in generating workflows that should produce correct answers but fail due to tool errors. This suggests that DWIM has a better understanding of each tool. Besides, DWIM is less likely to misuse tools when constructing workflows after training. Overall, these findings demonstrate that DWIM significantly enhances tool awareness.

12. Tool Library and Functionality

In this section, we introduce the details of the task-specific tool library (Table 8), including the functionalities of each tool and their corresponding models. Table 9 provides a comprehensive overview of the tools included in the proposed tool library and their respective functionalities. The table is structured to showcase the capabilities of each tool across different models (LLaVA-1.5-7B [37], BLIP2-Flan-T5-XXL [35], GPT-4o [22], and GroundingDINO-Base [39]) and provides a brief description of their specific functionalities.

- **Detector:** This functionality, supported by GroundingDINO, focuses on detecting objects within an image.
- **Check Existence:** GroundingDINO is also capable of checking the existence of specific objects within a given scene, contributing to basic visual verification tasks.
- **Simple Query Answer:** Both LLaVa-1.5 and BLIP2 excel in answering simple questions using a single word or phrase. This capability is valuable for tasks requiring concise and precise responses.
- **Complex Query Answer:** LLaVa-1.5 extends its capability to answering more complex questions, providing sentence-level responses that demand a deeper understanding of the image and associated context.
- **Captioning:** LLaVa-1.5 further supports image captioning, generating descriptive captions for input images to facilitate contextual interpretation.
- **Acquiring External Knowledge:** GPT-4o is the sole tool in this library designed to acquire external knowledge, which is essential for tasks that require external information beyond the given visual input.
- **Boolean to Yes/No:** This functionality would involve converting boolean values (True/False) into human-readable yes/no responses.
- **Image Crop:** This functionality is designed to crop images based on provided coordinates.
- **Property Matching:** It supports identifying the best-matching visual property among a set of options.
- **Verify Property:** It is capable of verifying visual properties.

13. Failure Case Analysis

While DWIM has achieved SoTA performance, there remains room for improvement in its design. In complex cases, as illustrated in Figure 9, DWIM may fail due to errors made by the LLMs, resulting in incorrect workflows or workflows that are logically correct but fail due to tool errors. In future iterations, we aim to enhance the ability of agentic LLMs to automatically select and utilize tools for better decision-making.

Furthermore, we investigate the primary limitations of current frozen LLMs when presented with 10-shot examples. Through human investigation of workflows leading to incorrect answers provided by frozen LLMs, we identified the following common issues: **lack of reasoning ability to determine when to stop**, **lack of self-correction ability**, and **lack of tool awareness**, meaning the proposed methods are logically correct but practically flawed.

14. Computational Costs

Running on four RTX A6000 GPUs, the average inference and training time per sample (in seconds) is as follows: DWIM (9.4, 14.4), HYDRA [27] (3.6, 28.8), and VisRep [30] (7.2, 7.2). HYDRA uses DQN for training, which is difficult to parallelize due to time constraints, and its official code does not support multi-GPU acceleration. Therefore, HYDRA training was conducted on a single RTX A6000 GPU.

To explore more computation information, we computed the average token count per sample for LLM of each method as shown in Table 11. Our method incurs slightly more computation than VisRep but achieves significantly better performance, while requiring far less than HYDRA (which uses GPT) and still outperforming it.

Table 11. Average Input and Output Token Counts of the LLM.

	DWIM (Ours)	VisRep (CVPR24)	HYDRA (ECCV24)
Avg. Tokens (In+Out)	5931.87	3520.44	9387.24

15. Prompt Template

In DWIM, the agentic LLM can autonomously explore the environment through three types of actions, as outlined in Section 3. In this section, we are providing both the prompt template for agent auto-exploration and the Python interface code enabling the agent’s perception capabilities.

Prompt 15.1: Auto-Exploring

Your job is to write code to solve questions about images. You have **access** to the `ImagePatch` class above.




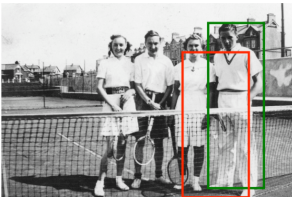
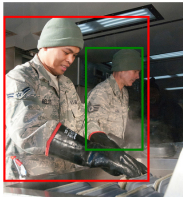
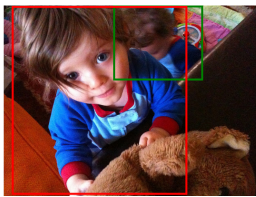
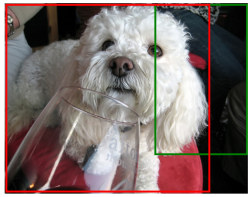
 <p>Query: How many people are wearing glasses?</p> <p>Answer: 2</p> <p>Ground truth: 3</p> <p>Main Issue: Tool Failures</p>	 <p>Query: Is the chair to the right or to the left of the person that stands by the wall?</p> <p>Answer: No</p> <p>Ground truth: Right</p> <p>Main Issue: Incorrect Workflow</p>	 <p>Query: Verify 'An umbrella that is sitting underneath a light'</p> <p>Answer: YES</p> <p>Ground truth: NO</p> <p>Main Issue: Incorrect Workflow</p>	 <p>Query: Who can use this transportation type?</p> <p>Answer: Anyone</p> <p>Ground truth: [Human, Student, People]</p> <p>Main Issue: Incorrect Evaluation</p>
 <p>Query: Find 'person on right'</p> <p>Answer: <input type="checkbox"/> bounding box</p> <p>Ground truth: <input type="checkbox"/> bounding box</p> <p>Main Issue: Tool Failures</p>	 <p>Query: Find 'man in uniform on right'</p> <p>Answer: <input type="checkbox"/> bounding box</p> <p>Ground truth: <input type="checkbox"/> bounding box</p> <p>Main Issue: Tool Failures</p>	 <p>Query: Find 'boy on right'</p> <p>Answer: <input type="checkbox"/> bounding box</p> <p>Ground truth: <input type="checkbox"/> bounding box</p> <p>Main Issue: Tool Failures</p>	 <p>Query: Find 'jeans behind dog'</p> <p>Answer: <input type="checkbox"/> bounding box</p> <p>Ground truth: <input type="checkbox"/> bounding box</p> <p>Main Issue: Incorrect Workflow</p>

Figure 9. Failure Case Analysis. Queries are presented in blue boxes, DWIM's answers are displayed in red boxes, and ground truth labels are shown in green boxes. Additionally, we provide the main issues causing DWIM to fail in completing the task in yellow boxes.

You will be able to interact with a Jupyter notebook. You have to carefully **format** your responses according to the following rules.

1. When you want to write code, you must use triple backticks inside a '<code>' tag.
2. When you want to **return** text you must use the '<thought>' tag. Example: '<thought>I think this **is** the answer.</thought>'
3. When you are done, you must use the '<done>' tag with no content inside. Example: '<done></done>'
4. The response **from** the notebook will be enclosed inside a '<result>' tag. Example: '<result>2</result>'
5. The image will be loaded **for** you **in** a variable called 'image', the image detail captioning will be provided.
6. If you can directly answer the question using a single word **or** phrase, Your final answer should be stored **in** a variable called 'final_answer'.

7. If you need more information, you can write code to get more information **from** image.
8. In each step, you can only use a `_single_` action.
9. Take care to indent multi-line code carefully, **and** think step by step to solve the problem incrementally.
10. Answer the question using a single word **or** phrase **and** store the answer **in** 'final_answer', then exit the task with a '<done>' tag.
11. You must provide a solution, **and** please do **not** refuse to answer even **if** you are **not** completely sure.
12. If 'final_answer' is 'True' **or** 'False', please use 'bool_to_yn' to convert it to 'yes' **or** 'no'.

Prompt 15.2: Python Code for ImagePatch Class

```
class ImagePatch:
    def __init__(self, image, left=None, lower=None, right=None, upper=None):
        self.image
        pass

    @property
    def area(self):
        pass

    def find(self, object_name):
        pass

    def exists(self, object_name):
        pass

    def verify_property(self, object_name, visual_property):
        pass

    def best_description_from_options(self, object_name, property_list):
        pass

    def simple_query(self, question):
        pass

    def crop_left_of_bbox(self, left, upper, right, lower):
        pass

    def crop_right_of_bbox(self, left, upper, right, lower):
        pass

    def crop_below_bbox(self, left, upper, right, lower):
        pass

    def crop_above_bbox(self, left, upper, right, lower):
        pass

    def llm_query(self, question):
        pass

def bool_to_ynsno(bool_answer: bool) -> str:
    pass
```