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

Fucai Ke[1,2] Vijay Kumar B G[3] Xingjian Leng[4] Zhixi Cai[2] Zaid Khan[5]
Weiqing Wang[2] Pari Delir Haghighi[2] Hamid Rezatofighi[2] Manmohan Chandraker[3,6]
[1]Building 4.0 CRC [2]Monash University [3]NEC Labs America [4]ANU [5]UNC Chapel Hill [6]UC San Diego
https://pokerme7777.github.io/DWIM.github.io/

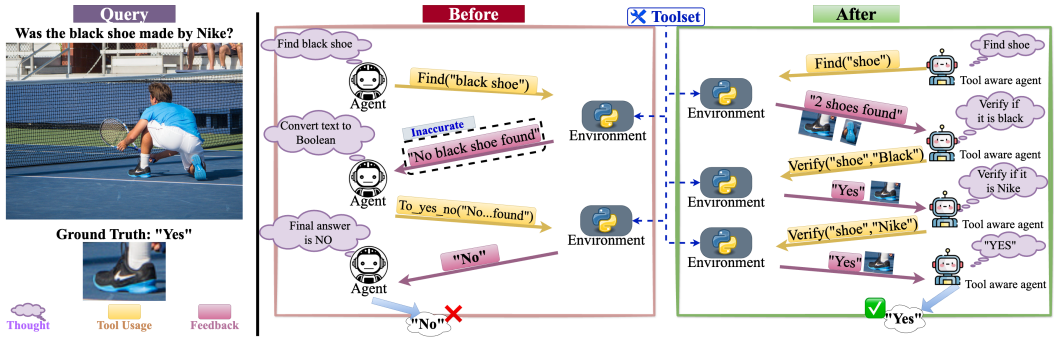


Figure 1. Comparison of the existing agent (Before) and our tool-aware agent (After). Both follow logically valid workflows with the same toolset, but our method improves tool selection and usage, minimizing tool-induced errors and ensuring more accurate, efficient execution.

Abstract

Visual reasoning (VR), which is crucial in many fields for enabling human-like visual understanding, remains highly challenging. Recently, compositional visual reasoning approaches, which leverage the reasoning abilities of large language models (LLMs) with integrated tools to solve problems, have shown promise as more effective strategies than end-to-end VR methods. However, these approaches face limitations, as frozen LLMs lack tool awareness in VR, leading to performance bottlenecks. While leveraging LLMs for reasoning is widely used in other domains, they are not directly applicable to VR due to limited training data, imperfect tools that introduce errors and reduce data collection efficiency in VR, and challenging in fine-tuning on noisy workflows. To address these challenges, we propose DWIM: i) Discrepancy-aware training Workflow generation, which assesses tool usage and extracts more viable workflows for training; and ii) Instruct-Masking fine-tuning, which guides the model to only clone effective actions, enabling the generation of more practical solutions. Our experiments demonstrate that DWIM achieves state-of-the-art performance across various VR tasks, exhibiting strong generalization on multiple widely-used datasets.

1. Introduction

This paper addresses the problem of visual reasoning (VR), which involves constructing detailed visual scene representation and reasoning about it in steps, similar to human cognition. VR involves interpreting and analyzing visual information in response to textual queries or prompts [3, 6, 27, 29] and encompasses a diverse range of tasks, including, but not limited to, visual commonsense reasoning [21], external-knowledge visual question answering [44, 51], vision-language compositionality understanding [17, 55], visual adversarial sample answering [34] visual grounding [25] and complex counting [1]. Beyond basic perception, VR enhances data interpretation by integrating reasoning with visual understanding, making it crucial in fields such as multimodal learning, autonomous driving, unmanned systems, and medical image analysis [6, 10, 11, 71]. However, despite its significant progress, VR remains highly challenging due to the inherent complexity of reasoning and the vast diversity of tasks [22, 35, 37, 73].

The most common VR solutions rely on end-to-end vision-language models [37, 38], which require large annotated datasets and significant computational resources, limiting their scalability and efficiency. Recently, compo-

sitional approaches, also known as workflow-based [62, 69] or tool-aware methods [49, 57], have emerged, leveraging frozen LLMs as planners to break down complex tasks into smaller, more manageable sub-tasks. By generating structured tool-utilization workflows and precise actions, these methods improve performance [6, 15, 27, 29, 41, 42, 52, 54]. However, their effectiveness remains limited, as most rely solely on the pre-trained knowledge of frozen LLMs.

Notably, the reasoning capabilities of LLMs have also been leveraged in other domains, such as mathematical reasoning, text-based games, and question answering, through similar step-by-step reasoning methods [14, 18, 30, 38, 48, 58, 65, 66, 73, 74]. In these domains, LLM agents are fine-tuned as domain experts rather than used in their frozen state. This is because frozen models are not explicitly trained to understand tool capabilities or generate effective tool-utilization workflows. To address this, approaches typically rely on large datasets containing annotated workflows, or leverage the inherent reasoning abilities of LLMs to generate workflows, selecting those that produce correct outcomes for training [53, 61, 64, 66]. LLMs are then trained on these workflows using supervised fine-tuning (SFT) or reinforcement learning (RL) to enhance their reasoning capabilities (*e.g.* [66, 70, 74]).

Similarly, while training LLMs is crucial for VR [30], applying the ideas from the aforementioned domains naively are not suitable due to the broader range of VR tasks and the relatively smaller dataset sizes. Consequently, most compositional VR approaches (*e.g.* [15, 27, 54, 67]) avoid training LLMs and instead rely on frozen language models. However, frozen LLMs limit overall performance [30]. Additionally, in the previously mentioned domains, action execution is generally reliable, as tools (*e.g.*, calculators, in-game functions, and search engines) operate based on deterministic algorithms. However, compositional visual reasoning heavily relies on deep learning models, which are more prone to inaccuracies and factual errors during execution. As illustrated in Figure 2, many logically sound workflows should yield correct answers (Desired block), but environmental feedback (*i.e.*, tool errors) cause some to fail in practice (Actual block). This limits the generation of valid workflows, worsening the data shortage for training, as faced by [30]. Additionally, filtering workflows solely based on the final outcome and cloning entire workflows during training, without identifying tool usage effectiveness, may cause the model to learn noise (*i.e.*, ineffective tool usage) in naive SFT.

In this paper, we introduce DWIM, which consists of: (i) Discrepancy-aware training Workflow generation to assess tool usage, discover more effective workflows, and optimize data utilization; and (ii) Instruct-Masking, a fine-tuning method that enhances tool awareness, improving task planning, and learns effective tool usage from noisy workflows.

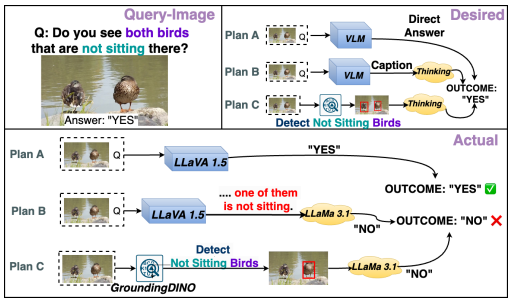


Figure 2. Tools are not always reliable and may occasionally provide incorrect information. Consequently, workflows expected to yield correct answers may fail due to tool-related inaccuracies.

Our discrepancy-aware training workflow generation, integrated with an agentic framework, enables LLMs to recognize ineffective tool usage by detecting factual errors or unexpected information in environmental feedback. By conditioning on the "answer," the LLM recognizes discrepancies (*e.g.*, factual errors) between execution feedback and expected outputs, describes them, and refines tool usage in following steps. This refinement enables the agent to explore alternative tool usages, which increase workflow success rates and ultimately yield more workflows for training.

Once workflow generation is complete, we fine-tune the LLM using instruct-masking. Treating a workflow as a sequence of actions (*e.g.*, task planning in text and tool usage in Python), we iteratively mask effective actions (*e.g.*, tool usage) at the semantic level based on prior effectiveness assessments and instruct the model to predict them. Ineffective tool usage and its corresponding next step (*i.e.*, discrepancy description and refinement thought) in the unmasked parts help the model recognize failed tool usage cases, while masking facilitates workflow augmentation. Meanwhile, instruction ensures the LLM clones only effective actions rather than blindly clone every step, as in naive SFT.

We evaluate our proposed methods on several popular VR datasets and compare with recent models, showing state-of-the-art performance. In summary, the key contributions of this work are as follows:

1. We introduce *discrepancy-aware training workflow generation*, the first known method to assess VR tool usage effectiveness, refine actions with alternatives, and increase workflow success rates, thereby enhancing training data utilization.
2. We propose an *instruct-masking fine-tuning*, which helps the model understand tool failures by exposing failure cases in the unmasked parts while instructing it to learn effective tool usage in masked part, expanding the trainable workflow size and enhancing tool awareness.
3. We conduct extensive experiments to demonstrate the advantages of our proposed method, highlighting its potential to operate effectively with reduced dependency on human prompt engineering.

Table 1. Difference between our work and similar compositional VRs.

	Supervision	Multi-turn Framework	Workflow Refinement	Improves LLM	Assess Tool Usage in Training	Improve Data Utilization
DWIM(ours)	YES	YES	YES	YES	YES	YES
HYDRA [27]	YES	YES	YES	NO	NO	NO
VisRep [30]	YES	NO	NO	YES	NO	NO
ViperGPT [54]	NO	NO	NO	NO	-	-
VisProg [15]	NO	NO	NO	NO	-	-
IdealGPT [15]	NO	NO	NO	NO	-	-
Chameleon [41]	NO	NO	NO	NO	-	-

2. Related Work

2.1. Compositional Visual Reasoning Approach

The compositional visual reasoning approach leverages powerful LLMs to solve visual reasoning tasks [7, 15, 27, 30, 43, 54, 67]. In this strategy, LLMs function as planners, reasoners or program generators to break down complex tasks into manageable subtasks and generate corresponding action plans through their chain-of-thought reasoning abilities [19, 24, 45]. The action plan may consist of high-level text descriptions outlining the task goal or fine-grained actions in text, symbols, or even Python code for logical operations [5, 9, 20, 31]. Based on the generated action plan, execution is carried out by leveraging various perception models or external tools, treating other end-to-end (E2E) models as tools (*e.g.*, [4, 36, 38, 39]) and integrating them into the tool library. For example, VisProg [15] and ViperGPT [54] leverage GPT models [5] to generate code for perception, question answering, and task-solving. Chameleon [41] uses GPT-3.5 [5] to generate a tool utilization sequence based on a provided template for task execution, while IdealGPT [67] employs a vision-language model to answer each sub-question generated by GPT-3.5, subsequently aggregating the results through reasoning.

However, these compositional methods face two key limitations: (1) they rely on frozen LLMs, which lack the ability to decompose problems into solvable subproblems that effectively leverage the provided tools. In-context learning examples alone are insufficient for enabling LLMs to fully grasp these tools' capabilities to complete tasks [48, 63]; and (2) they employ a single-turn framework instead of a multi-turn agentic approach, which confines LLMs to initial planning without iterative feedback from the environment, thereby limiting overall performance [27].

Recently, VisRep [30] introduced a method that leverages existing annotations to generate a coarse reward signal for visual reasoning tasks. It uses naive SFT to improve the LLM's visual program synthesis ability. However, VisRep's performance is limited by its single-turn framework, training workflow generation, and fine-tuning approach, which collects only a small portion of training workflows and clones full workflows, including ineffective actions. In contrast, another recent SOTA method, HYDRA [27], introduces a multi-turn incremental reasoning agentic framework. Its planner and reasoner modules use an LLM to generate instructions and executable code, while a RL agent makes high-level decisions based on feedback, allowing HYDRA to adjust actions for greater accuracy and effectiveness. However, as the planner itself is untrained, it may still produce incorrect next-step suggestions and lacks the ability to refine them. The differences between compositional VRs are illustrated in Table 1.

2.2. Workflow-based and Tool-aware LLM Agent

LLM agents offer advantages in affordability, transparency, and reproducibility for complex tasks [48, 63, 65, 66, 72]. Incremental reasoning agent framework such as [32, 47, 48, 59, 63, 65, 66, 68, 74], apply workflow-based and tool-aware LLM agents across language understanding, math, and beyond. To preparing for workflow training, these methods reply existing datasets or generate logically sound workflows using GPT-4 [2] or other LLMs, discarding outputs that yield incorrect results. This process is referred to in this paper as the "**standard**" training workflow generation method. However, it is unsuitable for visual reasoning tasks, as it fails to account for common errors arising from imprecise tools during interactions. Additionally, while their supervised fine-tuning approach is effective in certain domains, it may unintentionally reinforce incorrect tool usage in visual reasoning tasks.

2.3. Masking and Instruction Tuning

In BERT [28], masking refers to randomly replacing words or tokens in a sentence with a mask token (*e.g.*, `[MASK]`) and training the model to predict these hidden tokens based on surrounding context. Instruction tuning, on the other hand, is a fine-tuning process that trains LLMs to follow specific prompts using diverse instruction-output pairs. This method, which includes machine-generated instruction-following data, has been shown to enhance zero-shot performance across a range of tasks, especially in natural language processing and visual language models [5, 12, 28, 37, 46]. Our method, termed instruct-masking fine-tuning, is inspired by these approaches.

3. Methodology

To develop a more tool-aware LLM for visual reasoning, we propose DWIM: Discrepancy-aware training Workflow generation to assess tool usage, discover effective workflows, and optimize data utilization, along with Instruct-Masking fine-tuning to enhance tool awareness using collected noisy workflows.

3.1. Task Formulation

This subsection outlines the VR task formulation. In a visual reasoning task, let v be a visual input and q be a textual query about v, thus for each task $\xi = \{(v, q), y\}$, there exists a visual-textual query pair (v, q) corresponding to the answer y. When using the compositional visual reasoning method, there is a workflow $\boldsymbol{\omega} = \{\omega_{1:T}\}$ that contains each generated action ω from an agentic LLM π_θ. For any step t satisfying $1 \leq t \leq T$, we have:

$$\pi_\theta(\omega_t|e_0) = \prod_{t'=1}^{t} \pi_\theta(\omega_{t'}|e_{t'-1}), \tag{1}$$

$$e_t = \{e_{t-1}, \phi(\omega_t)\}, \tag{2}$$

where e_t is a combination of previous environmental feedback and the new feedback received after the interaction with action ω_t. ϕ represents the function that maps each action taken by the agent to the corresponding feedback from the current environment context. The environmental feedback e_t includes action execution outputs and error messages based on the query pair (v, q). The environment information for each task is given by $e_0 = \{(v, q), \delta\}$, where δ denotes the tool library documentation. Thus for each task, the answer prediction is approximately

$$\hat{y} := \phi(\omega_T) \tag{3}$$

$$\omega_T \sim \pi_\theta(\omega_T|e_{T-1}) \tag{4}$$

$$\pi_\theta(\omega_T|e_{T-1}) = \pi_\theta(\omega_T|\{e_0, \phi(\omega_1), \ldots, \phi(\omega_{T-1})\}) \tag{5}$$

For training, our objective is to optimize the parameters θ of the agentic LLM π to accurately generate the workflow. Building on these, we propose our workflow generation and fine-tuning methods.

3.2. Workflow Generation with Agentic Framework

This subsection introduces our proposed training workflow generation method for workflow collection, including discrepancy-aware recognition of ineffective tool usage, as well as the dynamic agentic framework designed for VR.

Our discrepancy-aware training workflow generation relies on the Auto-Exploring Agentic Framework, which is inspired by ReAct [65] and CodeAct [58]. However, unlike these methods, our proposed solution features a more interactive and dynamic agentic structure to enable discrepancy-aware recognition at each step, considering potential errors in environmental feedback during training workflow generation. This framework necessitates multi-turn interactions, moving beyond the traditional one-step planning framework (*e.g.*, [30, 54, 67]), as illustrated in Figure 5.

In a single-turn framework, $\hat{y}$ is approximated as:

$$\hat{y} := \phi(\omega_1) \tag{6}$$

$$\omega_1 \sim \pi_\theta(\omega_1|e_0), \tag{7}$$

Algorithm 1 Discrepancy-aware training workflow generation

Require: $v, q, \phi, \pi_\theta, y, D_{\boldsymbol{\omega}}$
$e \leftarrow (v, q); t \leftarrow 1; \boldsymbol{\omega} \leftarrow \{\}$ ▷ Initialization
while not final **do**
  $\omega \leftarrow \{\pi_\theta, \boldsymbol{\omega}, e, y\}$; $\boldsymbol{\omega}$.append(ω) ▷ Generate and store action
  $e \leftarrow \{\phi, \omega, e\}$ ▷ Update environment Info
  if discrepancy in (e, y) **then** ▷ Discrepancy-aware Recognition
    $\omega_{\text{Rethink}} \leftarrow \{\pi_\theta, \boldsymbol{\omega}, e, y\}$ ▷ Generate "Rethink"
    $\boldsymbol{\omega}$.append(ω_{Rethink}); $t \leftarrow t + 1$ ▷ Store "Rethink"
  end if
  $t \leftarrow t + 1$
end while
if $y = \hat{y} \leftarrow \phi(\omega_t)$ **then**
  $D_{\boldsymbol{\omega}}.append(\boldsymbol{\omega}\}$ ▷ Collect workflow
end if
return $D_{\boldsymbol{\omega}}$

where e_0 is the initial environment feedback. Thus, the prediction $\hat{y}$ is limited by π_θ and e_0 alone due to the single-turn setup. In contrast to existing single-turn agentic framework, multi-turn agentic framework [58, 65] offer the advantage of incorporating aggregated environment information $\{e_1, \ldots, e_{t-1}\}$ to enable incremental reasoning. These information e are grounded in the environmental context $\{(v, q), \delta\}$, enabling the model to iteratively refine its action generation process.

As illustrated in Figure 5 (Supplementary), there are three types of generated outputs inspired by CodeAct [58], $\omega_t \in \{\langle\text{Thought}\rangle, \langle\text{Code}\rangle, \langle\text{Done}\rangle\}$, corresponding to thinking, acting and sensing, and mission completion, respectively. The $\langle$Thought$\rangle$-type action ω_t enhances the reasoning process by analyzing the provided environmental information e_t to determine the optimal next step, either proceeding or refining the action, while articulating the rationale in natural language. When a $\langle$Code$\rangle$-type action (*i.e.*, tool usage) ω_t is generated, the agentic LLM explores $\phi(*)$, utilizing (perception) tools to gather additional necessary information. This new environmental feedback is appended to the existing environment information, updating e incrementally as $e_t = \{e_{t-1}, \phi(\omega_t)\}$ to support incremental reasoning. The exploration process is achieved by generating Python code that is executed using predefined Python APIs. Once e_{T-1} contains sufficient information and the agentic LLM has answered the query q, a $\langle$Done$\rangle$-type action ω_T is generated to conclude the task.

Discrepancy-aware Training Workflow Generation. Leveraging the agentic framework, we design a new training workflow generation method to enhance workflow creation and action assessment, with the algorithm presented in Algorithm 1. During the training workflow generation phase, the answer y is provided as prior knowledge, which modifies the policy π_θ as follows:

$$\pi_\theta(\omega_t|e_0, y) = \prod_{t'=1}^{t} \pi_\theta(\omega_{t'}|e_{t'-1}, y). \tag{8}$$

The **"standard"** workflow generation method, as dis-

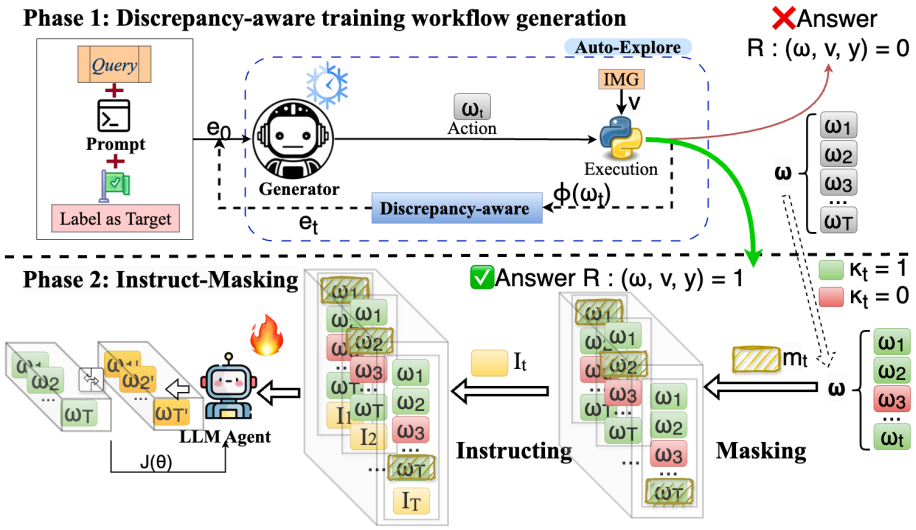


Figure 3. Overview of discrepancy-aware training workflow generation and instruct-masking process

cussed in Section 2.2, generates a workflow using $\pi_\theta(\omega_T|e_{T-1})$, which only accepts e_{T-1} as a pre-condition without y. A multi-turn framework with T-turn interactions produces ω_T based on e_{T-1}, whereas a single-turn framework relies solely on e_0. Given that y is correct, a richer prior and consistent policy increase the likelihood of executing a correct action.

Simply put, if each e_t is potentially inaccurate, conditioning on y enables the LLM to recognize discrepancies between the ideal outcome and actual feedback. We prompt the LLM to assess each step's effectiveness by identifying such discrepancies (*e.g.*, factual errors from tool failures). If discrepancies are found, LLM generates a *"Rethink"* ⟨Thought⟩ action ω_{Rethink}, which provides a natural language description of the discrepancy and a suggested alternative as the next step. The red parts in Algorithm 1 highlight the advantage of the designed discrepancy-aware training workflow generation compared to the "standard" methods. Relying on new generation policy, we can then collect a dataset of workflow $D_{\boldsymbol{\omega}}$ with assessment result of each step action for instruct-masking fine-tuning.

3.3. Instruct-Masking Fine-tuning

Based on the collected workflows, we fine-tune the model on these noisy workflows using stepwise assessment results. Our goal is to use the collected dataset $D_{\boldsymbol{\omega}}$ to improve the policy π_θ by tuning the parameters θ. Existing methods [27, 30, 66] directly use a binary-valued reward function $R : (\boldsymbol{\omega}, v, y) \to \{0, 1\}$, which disregards the effectiveness of individual actions ω_t. However, our discrepancy-aware training workflow generation method can store intermediate feedback e and enables action flagging to identify effective actions, as detailed in Section 9. Each action ω_t is flagged as effective ($\kappa_t = 1$) or ineffective ($\kappa_t = 0$), resulting in a flagged action sequence.

Then each effective action is masked and the LLM is instructed to reproduce it iteratively. The loss is computed between the original and the LLM-generated action, allowing the model to focus on learning effective tool usage and planning strategies, rather than memorizing entire noisy workflows. In contrast to BERT's masking [28], we mask the action in a workflow at the semantic level instead of token level, as illustrated in phase 2 of Figure 3. The instruct approach is inspired by instruction-tuning [5, 12, 38] and incorporates the concept of an end-of-sequence token [16, 26, 56, 60].

Instruct-Masking. We define m_t^ξ as a context-level mask for ω_t^ξ when $\kappa_t = 1$ in task ξ. The instruction I_t^ξ corresponds to instructing the model to regenerate the masked action ω_t instead of proceeding to the next step. We then transition $\boldsymbol{\omega}^\xi$ into $d_t^\xi = \{\omega_{1:t-1}^\xi, m_t^\xi, \omega_{t+1:T}^\xi, I_t^\xi\}$ and the new instruct-masking dataset can de denoted by D_M. Then, we apply behavioral cloning by minimize the reward-weighted loss

$$J(\theta) = \mathbb{E}_{(\omega,e)} \sim D_M[R(\boldsymbol{\omega}, v)\mathcal{L}_{\text{NLL}}(p, q; \theta)], \quad (9)$$

where $\mathcal{L}_{\text{NLL}}(p, q; \theta)$ is the negative log-likelihood loss

$$\mathcal{L}_{\text{NLL}}(p, q; \theta) = -\mathbb{E}_{(\omega,e)\sim D_M}\left[\sum_{t=1}^{T} \log \pi_\theta(\omega_t|d_t, e)\right]. \quad (10)$$

4. Experiments and Results

We evaluate DWIM on multiple datasets through extensive experiments, comparing it to SOTA. Experiments show that DWIM enhances LLM tool usage in both effectiveness and efficiency while demonstrating strong robustness.

Implementation Details. All experiments are conducted with 4 RTX A6000 GPUs. For all experiments requiring open-source LLMs, we use LLaMa-3.1-8B-Instruct as the LLM backbone [13]. We use vLLM [33] for accelerating the agent exploration process. To encourage the agent

to explore various tool-usage actions, a temperature of 0.8 is set for LLM generation. For model fine-tuning, we leverage Low-Rank-Adaptation (LoRA) [18] with a rank $r = 64$ and a scaling factor $\alpha = 16$. The training schedule uses a cosine annealing scheduler with a peak learning rate of 3×10^{-5} and a warm-up ratio of 0.05. The model is trained over 6 epochs with a global batch size of 128.

During evaluation, we apply greedy sampling for generative models to ensure the reproducibility of results. We strictly adhere to the evaluation protocols established by previous works [15, 27, 30, 54], including dataset splits and the official baseline codebase. The same tool library and language backbone were used, except for HYDRA, which employs GPT-4o due to the unavailability of GPT-3.5. Furthermore, to ensure optimal performance reproduction, we adjust the total number of in-context learning examples to 10 based on the provided examples in the official code. The choice of ten examples is supported by our study, which will be presented in Section 4.4 and Figure 4. As a result, the reported performance may vary slightly, being either higher or lower than the originally reported results in the HYDRA and VisRep papers.

Task. The visual reasoning task primarily encompasses visual commonsense reasoning (VCR), external-knowledge visual question answering (EKVQA), vision-language compositionality understanding (VLCU), visual adversarial samples answering (VASA), grounding detection (GD) and complex counting questions (CCQ).

Dataset and Metric. In the method comparison experiment, we train and test DWIM and baseline models separately across six datasets [1, 17, 21, 25, 34, 44], each representing a distinct task, to evaluate their performance. We evaluate generalization on four datasets to ensure fairness, as some require model-specific adaptations that could introduce bias. Each method follows a consistent training and evaluation framework using the same inference setup across training set [21] and all four evaluation sets [17, 44, 51, 55]. Following previous work [27, 30, 54], we use the same dataset split settings on TallyQA [1], which was not covered in prior studies, creating a train set of $1,000$ samples and test sets of 500 samples. Performance is assessed using accuracy (ACC) [27, 30, 41, 54, 67]. In the case of the VASA dataset (*i.e.* NaturalBench [34]), we report group-level ACC rather than single-question ACC, in alignment with the guidelines provided by the dataset authors. For the grounding detection task, we use intersection over union (IoU). Additionally, to assess the proportion of training data points that generate workflows yielding correct results and are suitable for training, we use the term "data utilization."

Tool Library in Different Task Environment. We follow previous works [15, 27, 30, 54, 67] on evaluation protocol using task-specific tool libraries for each task. These tool libraries might include BLIP2 [35], LLaVA-1.5 [37], GroundingDINO [39], which are integrated into all compositional visual reasoning methods for fair comparison. Our framework is also a plug-and-play system, meaning that the performance can be enhanced by incorporating more advanced perception tools. For details on tool utilization methods, please refer to the appendix.

4.1. Quantitative Analysis

Method Comparison Across Different Tasks: We compare our method with recent state-of-the-art (SOTA) compositional visual reasoning methods, including VisRep [30] and HYDRA [27], as shown in Table 2. *Frozen few-shot* refers to the method that uses a frozen LLaMa 3.1 within our designed inference framework. We report the average ACC/IoU score over three runs along with the standard deviation to ensure the reliability and consistency of our results. Both VisRep [30], HYDRA [27] and *Frozen few-shot* are evaluated with 10-shot in-context learning examples, as they rely heavily on in-context examples. In contrast, our method is evaluated without any in-context examples during the evaluation process because of our methods advantage. Additionally, we provide results for several end-to-end methods within the E2E block as reference points in Table 2, including the visual perception models used in all compositional approaches: **BLIP2** [35], **LLaVA-1.5** [37] and **GroundingDINO** [39].

Our method demonstrates SOTA performance across all tasks, achieving an average absolute improvement of over 9.73% compared to existing compositional approaches, and outperforming all utilized tools.

Generalization Ability Comparison: To evaluate generalization, we assess the cross-dataset performance of compositional approaches. Specifically, we use the checkpoint trained on GQA [21] to test models on datasets with the same inference setup (i.e., OKVQA [44], A-OKVQA [51], SugarCREPE [17], and Winoground [55]), ensuring a fair comparison.

As shown in Table 3, our method demonstrates strong generalization abilities, even without relying on in-context learning examples. Our method achieves an approximately 4.8% improvement compared to HYDRA [27] and VisRep [30], highlighting its effectiveness and strong zero-shot performance. Two E2E models, including LLaVa-1.5 [37] and BLIP2 [35], trained on datasets like GQA and OKVQA, serve as reference points. Our method achieves superior zero-shot cross-dataset performance, outperforming baselines and demonstrating capabilities comparable to GPT-4o.

Frozen LLMs encounter performance bottlenecks even as the number of in-context learning examples increases, compared to DWIM. We conducted experiments to demonstrate the robustness of our method compared to frozen LLMs with few-shot prompting. As illustrated in Figure 4, frozen agentic LLMs do not have ability to under-

Table 2. Comparison of average performance across six datasets, each corresponding to a distinct task. $\#Shots$ means the number of provided in-context learning examples. We use $*$ to highlight the second high score. For all E2E methods, we present their results in grey as they serve as reference points but are not compositional visual reasoning methods and are not intended as direct comparison targets.

			VCR	EKVQA	VLCU	VASA	GD	CCQ
Type	Method	$\#Shots$	GQA	OKVQA	SugarCREPE	NaturalBench	RefCOCO	TallyQA
E2E	BLIP2 [35]	-	45.5	31.4	53.0	2.9	-	49.0
	LLaVa-1.5 [37]	-	62.1	60.6	52.7	12.0	34.9	68.0
	GPT-4o [22]	-	58.5	33.4	62.5	16.8	30.5	76.4
	YOLO-world [8]	-	-	-	-	-	36.6	-
	GroundingDINO [39]	-	-	-	-	-	80.5	-
Compositional	Frozen few-shot	10	48.8 ± 2.2	56.2 ± 0.5	56.3 ± 0.9	4.8 ± 1.4	$63.0 \pm 1.8^*$	$60.2 \pm 0.5^*$
	VisRep [30]	10	51.4 ± 1.0	46.7 ± 2.0	$58.2 \pm 1.6^*$	$12.3 \pm 1.2^*$	55.2 ± 0.8	47.9 ± 1.2
	HYDRA [27]	10	$62.3 \pm 1.2^*$	$60.6 \pm 1.4^*$	55.5 ± 2.0	12.3 ± 1.9	60.4 ± 0.6	57.2 ± 2.3
	DWIM (Ours)	0	$\mathbf{69.3 \pm 1.0}$	$\mathbf{62.8 \pm 1.2}$	$\mathbf{74.6 \pm 1.3}$	$\mathbf{13.6 \pm 1.4}$	$\mathbf{82.7 \pm 0.9}$	$\mathbf{72.0 \pm 1.3}$
	Δ(abs) / Δ(%)	-	↑ **7.0** / **11.2**	↑ **2.2** / **3.6**	↑ **16.4** / **28.2**	↑ **1.3** / **10.6**	↑ **19.7** / **31.3**	↑ **11.8** / **19.6**

Table 3. Cross-dataset Generalization Ability Study. We use the checkpoint trained on the GQA training set to test models on other datasets. Results for all E2E methods and frozen agentic LLMs are shown in grey as reference points; these models are neither compositional nor trained and are not intended for direct comparison. Δ(abs) means absolute difference and Δ(%) means relative difference.

Type	Method	$\#Shots$	OKVQA (%)	A-OKVQA (%)	SugarCREPE (%)	Winoground (%)
E2E	BLIP2 [35]	-	31.4	41.0	53.0	49.9
	LLaVa-1.5 [37]	-	60.6	68.7	52.7	49.9
	GPT-4o [22]	-	33.4	63.2	62.5	65.6
Compositional	VisRep [30]	10	32.4 ± 2.0	63.5 ± 1.6	58.2 ± 1.9	49.3 ± 0.4
	HYDRA [27]	10	59.3 ± 1.0	62.1 ± 0.5	54.3 ± 1.0	50.1 ± 1.2
	DWIM (Ours on GQA)	0	$\mathbf{60.8 \pm 0.8}$	$\mathbf{69.8 \pm 1.3}$	$\mathbf{62.4 \pm 1.2}$	$\mathbf{57.4 \pm 0.1}$
	Δ(abs) / Δ(%)	–	↑ **1.5** / **2.5**	↑ **6.3** / **9.9**	↑ **4.2** / **7.2**	↑ **7.3** / **14.6**

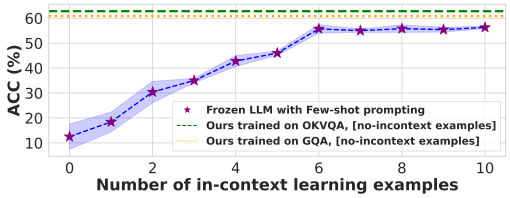


Figure 4. Frozen LLM with in-context learning example *v.s.* 0-shot trained LLM performance on OKVQA dataset.

Table 4. Comparison on average performance, TS: task-specific

Method	$\#Shots$	GQA (%)		SugarCREPE (%)	
		TS	Complete	TS	Complete
Frozen few-shot	10	48.8 ± 2.2	40.5 ± 1.4	56.3 ± 0.9	40.9 ± 0.4
VisRep	10	51.4 ± 1.0	41.7 ± 3.6	58.2 ± 1.6	46.4 ± 1.8
HYDRA	10	62.3 ± 1.2	53.3 ± 2.6	55.5 ± 2.0	43.6 ± 1.9
DWIM (Ours)	0	$\mathbf{69.3 \pm 1.0}$	$\mathbf{67.1 \pm 0.2}$	$\mathbf{74.6 \pm 1.3}$	$\mathbf{63.0 \pm 1.4}$
Δ(abs) / Δ(%)	-	↑ **7.0** / **11.2**	↑ **13.8** / **25.9**	↑ **16.4** / **28.2**	↑ **16.6** / **35.8**

stand provided tool and do not inherit with problem solving logic ability and only get around 12% ACC on OKVQA dataset. As the number of in-context learning examples increases, the performance of frozen LLMs improves but quickly reaches a saturation point. This suggests that further gains from extensive human prompt engineering may be marginal, as scaling up the number of examples yields diminishing returns. While examples can teach the LLM how to use tools, they cannot cover all possible situations where tools may fail. Overall, our method outperforms existing approaches, achieving significantly better performance with a small training dataset.

Dependence of Different Compositional Methods on Task-Specific Tool Libraries. All previous experiments follow the evaluation protocol, using task-specific tool libraries for each task. In this experiment, we train and evaluate all compositional methods using a complete tool library instead of a task-specific one on two tasks to examine their dependence on designed tool libraries. Results in Table 4 show that using a complete library decreases performance across all methods. This decline stems from the challenge of selecting optimal tools from a larger selection space, worsened by frozen LLMs' limited tool awareness. However, the greater performance disparity highlights DWIM's effectiveness and reduced reliance on manual adjustments.

Comparison of Tool-Use Efficiency Across Methods: We quantified the average tool utilization per successful sample to assess tool awareness from an efficiency perspective. Each tool use was counted as a single instance, yielding the following averages: Frozen few-shot: 2.87, HYDRA: 3.26, VisRep: 2.49, and DWIM: **1.74**. Lower values indicate better performance, highlighting DWIM's superior tool awareness compared to frozen LLMs. Further analysis of tool awareness is provided in the supplementary material.

4.2. Qualitative Analysis

We provide two qualitative analyses in the Appendix: one assessing DWIM's performance and another comparing training workflow generation.

Table 5. Ablation Study: Effectiveness of the proposed training workflow generation. W/O denotes the use of the "standard" (as described in Sec 3.2 Eq.5) method, while W indicates the use of the discrepancy-aware workflow generation strategy.

Dataset	Type	Generator	Data Utilization (%)	ACC (%)
GQA	W/O	LLaMa3.1(8B)	48.2	57.9
	W/O	GPT4O	50.9	58.0
	W	LLaMa3.1(8B)	**68.3**	**69.3**
		Δ(abs) / Δ(%)	↑ **17.4 / 34.2**	↑ **11.3 / 19.5**
SugarCREPE	W/O	LLaMa3.1(8B)	56.5	60.0
	W/O	GPT4O	59.3	60.7
	W	LLaMa3.1(8B)	**76.4**	**74.6**
		Δ(abs) / Δ(%)	↑ **17.1 / 28.8**	↑ **13.9 / 22.9**

4.3. Ablation Studies

In this section, we provide ablation studies on the key techniques proposed in DWIM to demonstrate their contributions to the final results.

Efficient Training Workflow Generation and Collection. In this study, we investigate the impact of discrepancy-aware training workflow generation on data utilization. Data utilization refers to the proportion of training data points that can generate workflows yielding correct results and are suitable for training. As shown in Table 5, we compare our method with the standard generation approach on the GQA and SugarCREPE datasets. The collected workflows are then used to fine-tune LLaMa3.1-8B, and performance is evaluated using Instruct Masking without in-context learning examples. On GQA, our approach improves data utilization by 17.4%, enabling more valid workflows and achieving an 11.3% absolute performance gain over the standard method. Similarly, on SugarCREPE, it increases data utilization by 17.1% and boosts absolute performance by 13.9%. This comparison highlights the effectiveness of our method in maximizing data utility and achieving optimal outcomes in visual reasoning tasks.

Instruct-Masking Fine-tuning. In this ablation study, we examine the effectiveness of the instruct-masking fine-tuning method compared to SFT on two tasks using GQA and SugarCREPE datasets as shown in Table 6. Leveraging workflows generated via the standard method, instruct-masking yields noticeable performance gains, with a 4.3% absolute performance improvement on the GQA dataset and a 2.9% absolute improvement on SugarCREPE. However, when paired with our discrepancy-aware training workflow generation and collection strategy, the absolute performance gains become even more pronounced, reaching 14.5% on GQA and 16.3% on SugarCREPE. Furthermore, instruct-masking exhibits a clear advantage over both *Random-Masking* and *Masking-W-Rethink*. Random-Masking indiscriminately masks actions rather than selectively targeting correct actions within the workflow, while Masking-W-Rethink masks both effective actions and ineffective actions that trigger "Rethink." These experimental results collectively demonstrate the effectiveness of instruct-masking in training on noisy workflows, leading to improved efficiency and accuracy in the training process.

Table 6. Ablation Study: Instruct-Masking Fine-tuning on GQA and SugarCREPE. "Random-Masking" refers to the process of randomly masking any action within a workflow and is shown in grey because it is neither an existing method nor our proposed method, but is included for the ablation study.

Fine-tune	Method	Model	GQA	SugarCREPE
Frozen	10-shot prompting	LLaMa3.1 (8B)	48.8	56.3
Frozen	10-shot prompting	GPT4o	51.0	57.7
SFT	Standard	LLaMa3.1 (8B)	53.6	57.1
Instruct-Masking	Standard	LLaMa3.1 (8B)	57.9	60.0
		Δ(abs) / Δ(%)	↑ **4.3 / 8.0**	↑ **2.9 / 5.1**
SFT	Discrepancy-aware	LLaMa3.1 (8B)	54.8	58.3
Random-Masking	Discrepancy-aware	LLaMa3.1 (8B)	65.1	63.5
Masking-W-Rethink	Discrepancy-aware	LLaMa3.1 (8B)	68.0	70.7
Instruct-Masking	Discrepancy-aware	LLaMa3.1 (8B)	69.3	74.6
		Δ(abs) / Δ(%)	↑ **14.5 / 26.5**	↑ **16.3 / 28.0**

Table 7. LLM Backbone Experiments on VCR Task.

Method	LLaMa-3.1-8B (%)	Mistral-v0.3-7B (%)	Mistral-v0.2-7B (%)
Frozen few-shot	48.8	52.0	45.1
DWIM (Ours)	69.3	67.5	63.7
Δ(abs) / Δ(%)	↑ **20.5 / 42.0**	↑ **15.5 / 29.8**	↑ **18.6 / 41.2**
Method	Qwen2.5-1.5B (%)	Qwen2.5-3B (%)	Qwen2.5-7B (%)
Frozen few-shot	18.7	41.4	53.4
DWIM (Ours)	48.2	63.0	65.8
Δ(abs) / Δ(%)	↑ **29.5 / 157.6**	↑ **21.6 / 52.2**	↑ **12.4 / 23.2**

Experiment Using Various LLM Backbones for DWIM. All previous results show that DWIM significantly enhances LLaMa-3.1's capabilities. To evaluate its generality, we apply DWIM to various open-source models (e.g., LLaMa-3.1-8B [40], Mistral-v0.2/0.3-7B [23], and three Qwen2.5 variants [50]), and compare their performance to frozen LLMs with 10-shot in-context learning. As shown in Table 7, DWIM improves absolute performance by over 12.4% across all models, demonstrating its effectiveness.

5. Conclusion

This paper introduces an efficient discrepancy-aware training workflow generation and instruct-masking fine-tuning method for tool-aware visual reasoning. These innovations address the challenges of efficiently generating workflow data within environments with potential incorrect feedback and fine-tuning on noisy workflows. DWIM enhances LLM tool awareness, enabling more accurate and efficient tool usage for complex visual reasoning tasks. Extensive experiments on public benchmarks demonstrate that our method achieves state-of-the-art performance, offering a more generalizable and robust solution for compositional visual reasoning tasks.

Acknowledgments

This work was supported by Building 4.0 CRC and the Commonwealth of Australia through the Cooperative Research Centres Program. It was also partially funded by the DARPA ANSR program (FA8750-23-2-1016) and the ARC DECRA program (DE250100032).

References

[1] Manoj Acharya, Kushal Kafle, and Christopher Kanan. Tallyqa: Answering complex counting questions. In *Proceedings of the AAAI conference on artificial intelligence*, pages 8076–8084, 2019. 1, 6

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 3

[3] Saeed Amizadeh, Hamid Palangi, Alex Polozov, Yichen Huang, and Kazuhito Koishida. Neuro-symbolic visual reasoning: Disentangling. In *International Conference on Machine Learning*, pages 279–290. PMLR, 2020. 1

[4] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*, 2023. 3

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 3, 5

[6] Zhixi Cai, Cristian Rojas Cardenas, Kevin Leo, Chenyuan Zhang, Kal Backman, Hanbing Li, Boying Li, Mahsa Ghorbanali, Stavya Datta, Lizhen Qu, et al. Neusis: A compositional neuro-symbolic framework for autonomous perception, reasoning, and planning in complex uav search missions. *arXiv preprint arXiv:2409.10196*, 2024. 1, 2

[7] Zhixi Cai, Fucai Ke, Simindokht Jahangard, Maria Garcia de la Banda, Reza Haffari, Peter J Stuckey, and Hamid Rezatofighi. Naver: A neuro-symbolic compositional automaton for visual grounding with explicit logic reasoning. *arXiv preprint arXiv:2502.00372*, 2025. 3

[8] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. Yolo-world: Real-time open-vocabulary object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16901–16911, 2024. 7

[9] 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. *Journal of Machine Learning Research*, 24(240): 1–113, 2023. 3

[10] Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, and Ziran Wang. Drive as you speak: Enabling human-like interaction with large language models in autonomous vehicles. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 902–909, 2024. 1

[11] Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, Yang Zhou, Kaizhao Liang, Jintai Chen, Juanwu Lu, Zichong Yang, Kuei-Da Liao, et al. A survey on multimodal large language models for autonomous driving. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 958–979, 2024. 1

[12] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. Instructblip: towards general-purpose vision-language models with instruction tuning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2024. 3, 5

[13] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 5

[14] Zhi Gao, Bofei Zhang, Pengxiang Li, Xiaojian Ma, Tao Yuan, Yue Fan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, and Qing Li. Multi-modal agent tuning: Building a vlm-driven agent for efficient tool usage. *https://arxiv.org/abs/2412.15606*, 2025. 2

[15] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023. 2, 3, 6

[16] Julia Hirschberg and Christopher D Manning. Advances in natural language processing. *Science*, 349(6245):261–266, 2015. 5

[17] Cheng-Yu Hsieh, Jieyu Zhang, Zixian Ma, Aniruddha Kembhavi, and Ranjay Krishna. Sugarcrepe: Fixing hackable benchmarks for vision-language compositionality. *Advances in neural information processing systems*, 36, 2024. 1, 6

[18] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. 2, 6

[19] Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. Visual program distillation: Distilling tools and programmatic reasoning into vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9590–9601, 2024. 3

[20] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022. 3

[21] Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6700–6709, 2019. 1, 6

[22] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024. 1, 7, 4

[23] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023. 8

[24] Xinyi Jiang, Guoming Wang, Junhao Guo, Juncheng Li, Wenqiao Zhang, Rongxing Lu, and Siliang Tang. Diem: Decomposition-integration enhancing multimodal insights. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27304–27313, 2024. 3

[25] Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara Berg. Referitgame: Referring to objects in photographs of natural scenes. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 787–798, 2014. 1, 6

[26] Fucai Ke, Weiqing Wang, Weicong Tan, Lan Du, Yuan Jin, Yujin Huang, and Hongzhi Yin. Hitskt: A hierarchical transformer model for session-aware knowledge tracing. *Knowledge-Based Systems*, 284:111300, 2024. 5

[27] Fucai Ke, Zhixi Cai, Simindokht Jahangard, Weiqing Wang, Pari Delir Haghighi, and Hamid Rezatofighi. Hydra: A hyper agent for dynamic compositional visual reasoning. In *European Conference on Computer Vision*, pages 132–149. Springer, 2025. 1, 2, 3, 5, 6, 7, 4

[28] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, page 2. Minneapolis, Minnesota, 2019. 3, 5

[29] Zaid Khan, Vijay Kumar B G, Samuel Schulter, Manmohan Chandraker, and Yun Fu. Exploring question decomposition for zero-shot vqa. In *Advances in Neural Information Processing Systems*, pages 56615–56627. Curran Associates, Inc., 2023. 1, 2

[30] Zaid Khan, Vijay Kumar BG, Samuel Schulter, Yun Fu, and Manmohan Chandraker. Self-training large language models for improved visual program synthesis with visual reinforcement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14344–14353, 2024. 2, 3, 4, 5, 6, 7

[31] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022. 3

[32] Jaywon Koo, Ziyan Yang, Paola Cascante-Bonilla, Baishakhi Ray, and Vicente Ordonez. Proptest: Automatic property testing for improved visual programming. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8241–8256, 2024. 3

[33] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023. 5

[34] Baiqi Li, Zhiqiu Lin, Wenxuan Peng, Jean de Dieu Nyandwi, Daniel Jiang, Zixian Ma, Simran Khanuja, Ranjay Krishna, Graham Neubig, and Deva Ramanan. Naturalbench: Evaluating vision-language models on natural adversarial samples. *Advances in Neural Information Processing Systems*, 37:17044–17068, 2025. 1, 6

[35] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023. 1, 6, 7, 4

[36] Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10965–10975, 2022. 3

[37] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26296–26306, 2024. 1, 3, 6, 7, 4

[38] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024. 1, 2, 3, 5

[39] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *European Conference on Computer Vision*, pages 38–55. Springer, 2024. 3, 6, 7, 4

[40] Introducing Llama. 3.1: Our most capable models to date. *Meta*, 2024. 8

[41] 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. *Advances in Neural Information Processing Systems*, 36, 2024. 2, 3, 6

[42] Ziyu Ma, Shutao Li, Bin Sun, Jianfei Cai, Zuxiang Long, and Fuyan Ma. Gerea: Question-aware prompt captions for knowledge-based visual question answering. *arXiv preprint arXiv:2402.02503*, 2024. 2

[43] Ziyu Ma, Chenhui Gou, Hengcan Shi, Bin Sun, Shutao Li, Hamid Rezatofighi, and Jianfei Cai. Drvideo: Document retrieval based long video understanding. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 18936–18946, 2025. 3

[44] Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Ok-vqa: A visual question answering benchmark requiring external knowledge. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 3195–3204, 2019. 1, 6

[45] Chancharik Mitra, Brandon Huang, Trevor Darrell, and Roei Herzig. Compositional chain-of-thought prompting for large multimodal models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14420–14431, 2024. 3

[46] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022. 3

[47] Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei

Huang, and Huajun Chen. Agent planning with world knowledge model. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. 3

[48] Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Huajun Chen, et al. Autoact: Automatic agent learning from scratch for qa via self-planning. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024. 2, 3

[49] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40, 2024. 2

[50] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. 8

[51] Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. A-okvqa: A benchmark for visual question answering using world knowledge. In *European conference on computer vision*, pages 146–162. Springer, 2022. 1, 6

[52] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[53] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[54] D'idac Sur'is, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11854–11864, 2023. 2, 3, 4, 6

[55] Tristan Thrush, Ryan Jiang, Max Bartolo, Amanpreet Singh, Adina Williams, Douwe Kiela, and Candace Ross. Winoground: Probing vision and language models for visio-linguistic compositionality. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5238–5248, 2022. 1, 6

[56] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017. 5

[57] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. 2

[58] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, pages 50208–50232. PMLR, 2024. 2, 4, 1

[59] Zora Zhiruo Wang, Graham Neubig, and Daniel Fried. Trove: inducing verifiable and efficient toolboxes for solving programmatic tasks. In *Proceedings of the 41st International Conference on Machine Learning*, pages 51177–51191, 2024. 3

[60] Ron J Weiss, Jan Chorowski, Navdeep Jaitly, Yonghui Wu, and Zhifeng Chen. Sequence-to-sequence models can directly translate foreign speech. *arXiv preprint arXiv:1703.08581*, 2017. 5

[61] Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis Ioannidis, Karthik Subbian, Jure Leskovec, and James Y Zou. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. *Advances in Neural Information Processing Systems*, 37:25981–26010, 2025. 2

[62] Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. Stateflow: Enhancing llm task-solving through state-driven workflows. In *First Conference on Language Modeling*, 2024. 2

[63] Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Watch every step! llm agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1556–1572, 2024. 3

[64] Yijun Yang, Tianyi Zhou, Kanxue Li, Dapeng Tao, Lusong Li, Li Shen, Xiaodong He, Jing Jiang, and Yuhui Shi. Embodied multi-modal agent trained by an llm from a parallel textworld. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26275–26285, 2024. 2

[65] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *ICLR*, 2022. 2, 3, 4, 1

[66] Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. Agent lumos: Unified and modular training for open-source language agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12380–12403, 2024. 2, 3, 5

[67] Haoxuan You, Rui Sun, Zhecan Wang, Long Chen, Gengyu Wang, Hammad Ayyubi, Kai-Wei Chang, and Shih-Fu Chang. Idealgpt: Iteratively decomposing vision and language reasoning via large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11289–11303, 2023. 2, 3, 4, 6

[68] Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi Fung, Hao Peng, and Heng Ji. Craft: Customizing llms by creating and retrieving from specialized toolsets. In *The Twelfth International Conference on Learning Representations*, 2024. 3

[69] Zhen Zeng, William Watson, Nicole Cho, Saba Rahimi, Shayleen Reynolds, Tucker Balch, and Manuela Veloso. Flowmind: automatic workflow generation with llms. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, pages 73–81, 2023. 2

[70] Simon Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Peter Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma,

et al. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. *Advances in Neural Information Processing Systems*, 37:110935–110971, 2025. 2

[71] Li-Ming Zhan, Bo Liu, Lu Fan, Jiaxin Chen, and Xiao-Ming Wu. Medical visual question answering via conditional reasoning. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 2345–2354, 2020. 1

[72] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 19632–19642, 2024. 3

[73] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. In *The Twelfth International Conference on Learning Representations*, 2024. 1, 2

[74] Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Ningyu Zhang, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, and Huajun Chen. Knowagent: Knowledge-augmented planning for llm-based agents. *arXiv preprint arXiv:2403.03101*, 2024. 2, 3

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

Supplementary Material

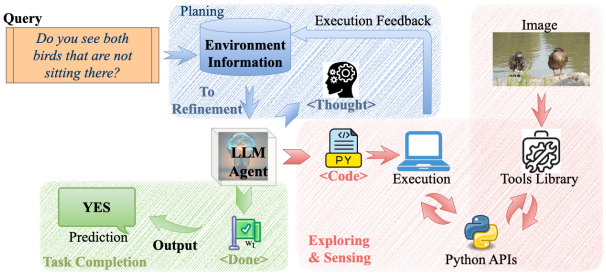


Figure 5. Auto-Exploring Agentic Framework. The LLM agent generates ⟨Code⟩ for execution, ⟨Thought⟩ for reasoning, or ⟨Done⟩ to complete the task. It dynamically generates or refines actions while storing environmental information for incremental reasoning.

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

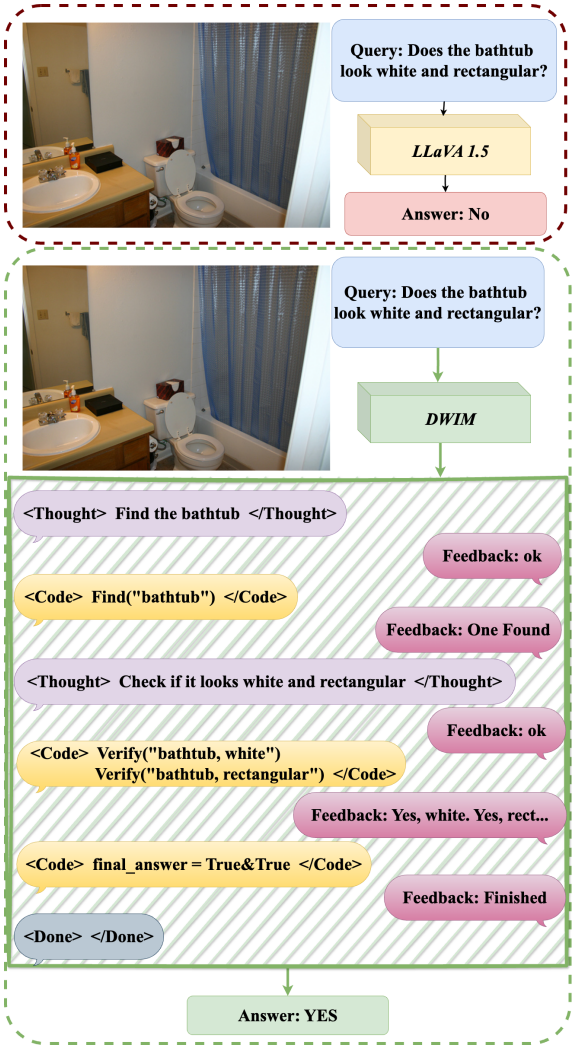


Figure 6. DWIM Qualitative Result Example

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.

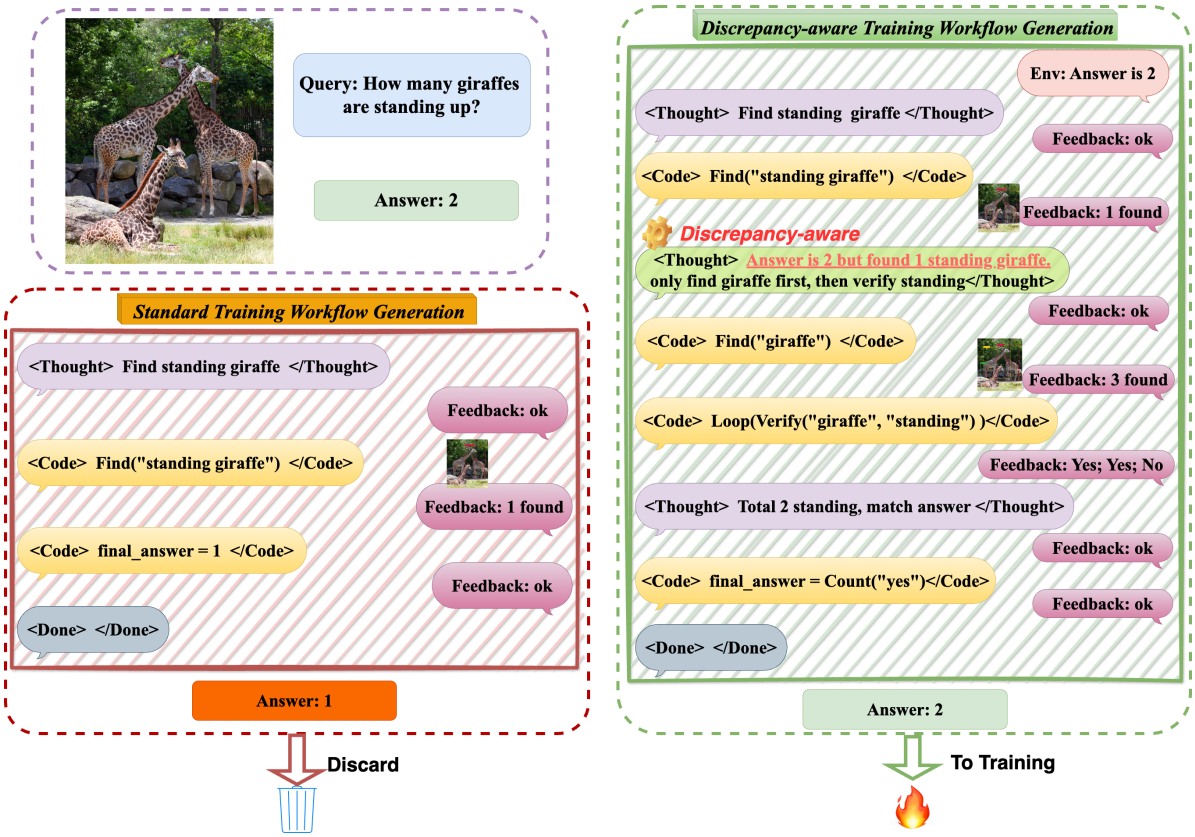


Figure 7. "Standard" *v.s.* "Discrepancy-aware training" Training Workflow Generation

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 ⟨Code⟩-actions, 23.5% are ⟨Thought⟩-actions, and 24.4% are ⟨Done⟩-actions.

In DWIM, any ⟨Code⟩-action flagged by environment feedback as "Traceback" is flagged as ineffective. Similarly, a action preceding a ⟨Thought⟩-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 ⟨Thought⟩-action. A total of 41 ineffective ⟨Code⟩-actions, 3 actions preceding a ⟨Thought⟩-action with the context "rethink," and 26 discrepancy-aware ⟨Thought⟩-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 ⟨Code⟩-actions. Additionally, 3 more actions triggering "rethink" or "replan" ⟨Thought⟩-actions and 2 additional discrepancy-aware ⟨Thought⟩ actions were identified.

As illustrated in Figure 8, all normal ineffective ⟨Code⟩-actions were flagged; however, only 50% of ineffective ⟨Thought⟩-action preceding a ⟨Thought⟩-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-

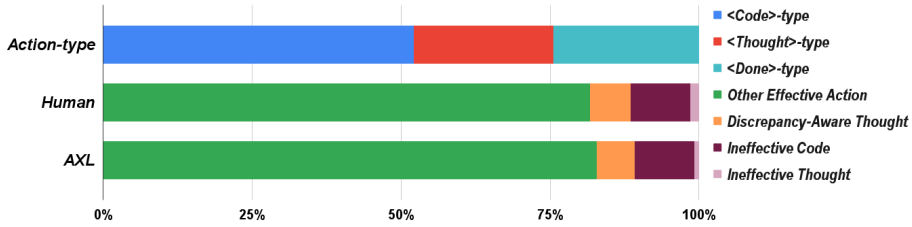


Figure 8. DWIM and Human Flagging of Ineffective Actions on Collected Workflows. DWIM's flagging results are close to those of human evaluators; however, there is still room for improvement, particularly in flagging actions that trigger "rethink."

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	✓	✓	✓	✓	✓	✓

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**

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 awareness 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.
```

Query: How many **people** are **wearing glasses**?

Answer: **2**

Ground truth: **3**

Main Issue: Tool Failures

Query: Is the **chair** to the **right** or to the **left** of the person that **stands by the wall**?

Answer: **No**

Ground truth: **Right**

Main Issue: Incorrect Workflow

Query: Verify 'An umbrella that is **sitting underneath** a light'

Answer: **YES**

Ground truth: **NO**

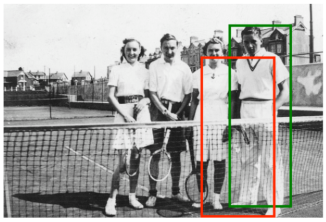
Main Issue: Incorrect Workflow

Query: **Who** can use this transportation type?

Answer: **Anyone**

Ground truth: **[Human, Student, People]**

Main Issue: Incorrect Evaluation

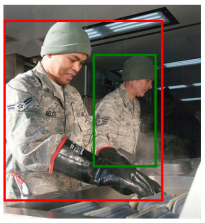


Query: Find 'person on right'

Answer: bounding box

Ground truth: bounding box

Main Issue: Tool Failures

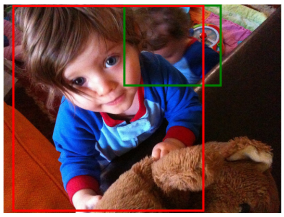


Query: Find 'man in uniform on right'

Answer: bounding box

Ground truth: bounding box

Main Issue: Tool Failures

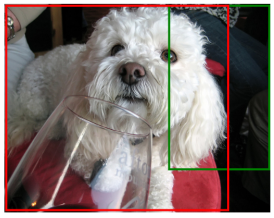


Query: Find 'boy on right'

Answer: bounding box

Ground truth: bounding box

Main Issue: Tool Failures



Query: Find '**jeans** behind dog'

Answer: bounding box

Ground truth: bounding box

Main Issue: Incorrect Workflow

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 detial 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_yesno' 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_yesno(bool_answer: bool) -> str:
    pass
```