VipAct: Visual-Perception Enhancement via Specialized VLM Agent Collaboration and Tool-use

Anonymous ACL submission

Abstract

Despite their strong performance in integrating textual and visual information, vision-language models (VLMs) still face challenges in fine-grained visual perception tasks that demand detailed pixel-level analysis and reasoning. We introduce VIPACT, an agent framework that enhances VLMs through multi-agent collaboration and vision expert models for precise visual understanding and reasoning. VIPACT features an orchestrator agent for task analysis, planning, and coordination, alongside specialized agents for tasks like image captioning and vision expert models for high-precision perception. This approach improves VLMs' fine-grained visual perception by integrating planning, reasoning, and tool use. We evaluate VIPACT on diverse visual perception benchmarks, showing significant improvements over state-of-the-art baselines across multiple VLMs. Ablation studies highlight the importance of multi-agent collaboration for detailed System-2 reasoning and the critical role of image input in task planning. Error analysis further reveals inherent VLM limitations, offering insights for future improvements.

1 Introduction

Recent advances in large multimodal models (LMMs), particularly vision-language models (VLMs) (OpenAI, 2024; Bai et al., 2023; Chen et al., 2024b), have shown impressive performance in integrating textual and visual information. Models like GPT-4o (OpenAI, 2024) achieve strong results on various image-text benchmarks (Hudson and Manning, 2019; Lu et al., 2023; Yue et al., 2024) and hold promise for real-world applications such as web navigation (Zheng et al., 2024a; He et al., 2024a). However, despite these advancements, studies (Rahmanzadehgervi et al., 2024; Fu et al., 2024; Tong et al., 2024; Li et al., 2024c) show that SOTA VLMs still struggle with fine-grained visual perception tasks, such as detecting line intersections or object boundaries—tasks that are trivial for humans. Overcoming these challenges is essential for deploying VLMs in critical applications like surgical robotics and autonomous driving, which demand precise visual understanding.

To address these challenges, prior works have explored visual programming methods (Subramanian et al., 2023; Hu et al., 2024b; Gupta and Kembhavi, 2023; Surís et al., 2023; Mialon et al., 2023; Wu et al., 2023a; Yang et al., 2023c), where text queries are input into LLMs to generate code that invokes vision-specific models, using their outputs directly as predictions. While effective for predefined tasks, these methods lack generalizability beyond existing toolsets, limiting their use as universal visual perception solutions. Another line of research focuses on prompting strategies to elicit foundation models' System-2 reasoning by involving iterative reasoning with intermediate tokens (Yu et al., 2024; Saha et al., 2024). Textual prompting methods (Wei et al., 2022; Saha et al., 2023; Yao et al., 2024; Besta et al., 2024) elicit LLMs to generate structured reasoning steps for complex text-based tasks, but their efficacy on fine-grained visual perception is underexplored. Similarly, visual prompting techniques (Lei et al., 2024; Yang et al., 2023a; Wu et al., 2024), which add artifacts like bounding boxes or masks to images, guide VLMs in interpreting visual data. While promising for some compositional visual reasoning, it is still unclear whether VLMs can accurately perceive such visual prompts, let alone whether these methods improve performance in visual perception.

To fill this gap, and inspired by advances in LLM-based agents (Wang et al., 2024d; Liu et al., 2023b; Significant-Gravitas, 2024; Wang et al., 2024a; Shen et al., 2024), we propose VIPACT (**VI**sual-**P**erception via VLM **A**gent **C**ollaboration and **T**ool-use), a general VLM-based framework that integrates multi-agent collaboration and vision expert models for fine-grained visual perception tasks. As shown in Figure 1, VIPACT consists of three core components: (1) an **orchestrator agent** that manages the workflow by analyzing tasks, coordinating agents, selecting tools, summarizing evidence, and deducing final answers; (2) **specialized agents** for tasks such as image captioning, visual prompt description, and image comparison, providing detailed visual analysis to the orchestrator; and (3) **vision expert models**, offering task-specific, fine-grained perceptual information to address VLMs' limitations. We evaluate VIPACT against SOTA baselines across benchmarks that include diverse visual perception tasks featuring complex elements like visual prompts and multi-image inputs. VIPACT consistently outperforms previous baselines on all tasks with different VLMs. Besides, our in-depth analysis highlights the importance of multi-agent collaboration in eliciting more detailed System-2 reasoning, as well as the critical role of visual input for task planning, with improved error handling and evidence aggregation.

Our key contributions are as follows: (1) VIPACT, a multi-modal agent framework that synergizes multi-agent collaboration with vision expert models to enhance fine-grained visual perception. It is an autonomous system capable of handling diverse visual perception tasks using a single prompt template. It leverages a VLM for task analysis, planning, and invoking multi-agent collaboration, with flexible plug-and-play modular components. (2) We conduct experiments across diverse visual perception benchmarks, demonstrating VIPACT's advantages over SOTA baselines; (3) We systematically analyze previous methods that proved to be effective in improving the general capabilities of foundation models for fine-grained visual perception, revealing their inconsistent effectiveness. (4) We present comprehensive ablation studies to assess the impact of multi-agent collaboration, visual input for planning, and each component of VIPACT, along with a detailed error analysis identifying the limitations of current VLMs, which serve as bottlenecks for further improvement.

2 Related Work

VLM-based Agent. Advancements in LLM capabilities like task decomposition and instruction following have spurred the development of LLM-based agents across diverse applications (Zhang et al., 2023c; Xi et al., 2023; Chen et al., 2023a; Significant-Gravitas, 2024; Shen et al., 2024; Deng et al., 2024a; Zhang et al., 2024e; Xie et al., 2024a; Liu et al., 2023b,a; Zhang et al., 2023a; Zhou et al., 2023). The introduction of visually capable models like GPT-4o (OpenAI, 2024) and other VLMs (Shi et al., 2024b) has positioned them as backbones for vision-centric agents (Hu et al., 2024a). Current research largely focuses on Web/GUI agents for interface interaction (Yan et al., 2023; Yang et al., 2023b; Zheng et al., 2024a; Xie et al., 2024c; Kapoor et al., 2024; Zhang et al., 2024a; Koh et al., 2024; Wang et al., 2024c; Lù et al., 2024; Zhang et al., 2024b; Deng et al., 2024b; You et al., 2024; Zheng et al., 2024b; Fan et al., 2024; Wang et al., 2024b; He et al., 2024b) and embodied agents controlling robots (Nasiriany et al., 2024; Tan et al., 2024; Ma et al., 2024; Xie et al., 2024b; Yang et al., 2024b; Szot et al., 2024). However, VLM-based agents specifically for natural image perception tasks remain unexplored.

Visual Programming. Recent LLMs excel at code generation (Gao et al., 2023; Zhang et al., 2023c, 2024e,d; Schick et al., 2024), enabling them to solve reasoning tasks via tool use and reduce hallucinations, especially in areas like mathematical reasoning (Cobbe et al., 2021; Hendrycks et al., 2021). This paradigm has been extended to vision tasks (Subramanian et al., 2023; Hu et al., 2024b; Gupta and Kembhavi, 2023; Surís et al., 2023; Mialon et al., 2023; Wu et al., 2023a; Koo et al., 2024). Systems like MM-REACT (Yang et al., 2023c) integrate LLMs with vision experts following ReAct's (Yao et al., 2023) prompt template, while ViperGPT (Surís et al., 2023) and VisProg (Gupta and Kembhavi, 2023) leverage LLMs to generate executable Python code for visual reasoning without extra training. However, these methods often depend solely on text queries for code generation and employ rigid tool selection, hindering adaptation to new tasks. This typically limits their application to simpler visual question answering scenarios (Hudson and Manning, 2019; Suhr et al., 2019; Marino et al., 2019), lacking support for fine-grained perception, visual prompts, or multi-image inputs, thereby restricting their utility in more complex visual reasoning. Table 1 provides a detailed comparison.

3 VipAct Framework

Our proposed framework, **VIPACT**, is illustrated in Figure 1. VIPACT consists of three main components: (1) **orchestrator agent** (Section 3.1),

Methods	Reas.	Tool	Multi-Ag.	Plan Img	Exec Img	Img Loop	Multi-Img	Vis. Prompt
ReAct (Yao et al., 2023)	✓	✓	✗	✗	✗	✗	✗	✗
MM-ReAct (Yang et al., 2023c)	✓	✓	✗	✗	✓	✗	✗	✗
ViperGPT (Surís et al., 2023)	✗	✓	✗	✗	✓	✗	✗	✗
VisProg (Gupta and Kembhavi, 2023)	✗	✓	✗	✗	✓	✗	✗	✗
CodeVQA (Subramanian et al., 2023)	✗	✓	✗	✗	✓	✗	✗	✗
VIPACT (Ours)	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison of VIPACT with other LLM/VLM-based agentic frameworks. ✓ indicates the presence of a specific feature in the corresponding framework, ✗ its absence. Column abbreviations: "Reas." for modules to elicit reasoning process, "Tool." for tool integration, "Multi-Ag." for multi-agent support, "Plan Img" for image input in planning, "Exec Img" for image input in execution, "Img Loop" for image use in iterative loops, "Multi-Img" for multi-image support, and "Vis. Prompt" for specific design for images containing visual prompts.

which controls the entire workflow by analyzing task requirements and task plans, initiating collaboration with other agents, selecting appropriate vision expert models, summarizing evidence from other agents or tools, and deducing the final answer. (2) **specialized agents** (Section 3.2), designed to handle specific tasks such as image captioning, visual prompt description, and image comparison. These agents provide detailed information to the orchestrator agent. (3) **vision expert models** (Section 3.3), which include specialized task-specific vision models that provide accurate, fine-grained perceptual information, addressing limitations of current VLMs. Intuitively, VIPACT enhances the VLM's System-2 reasoning by generating detailed intermediate reasoning steps through multi-agent collaboration while leveraging the high-precision perceptual information from vision expert models.

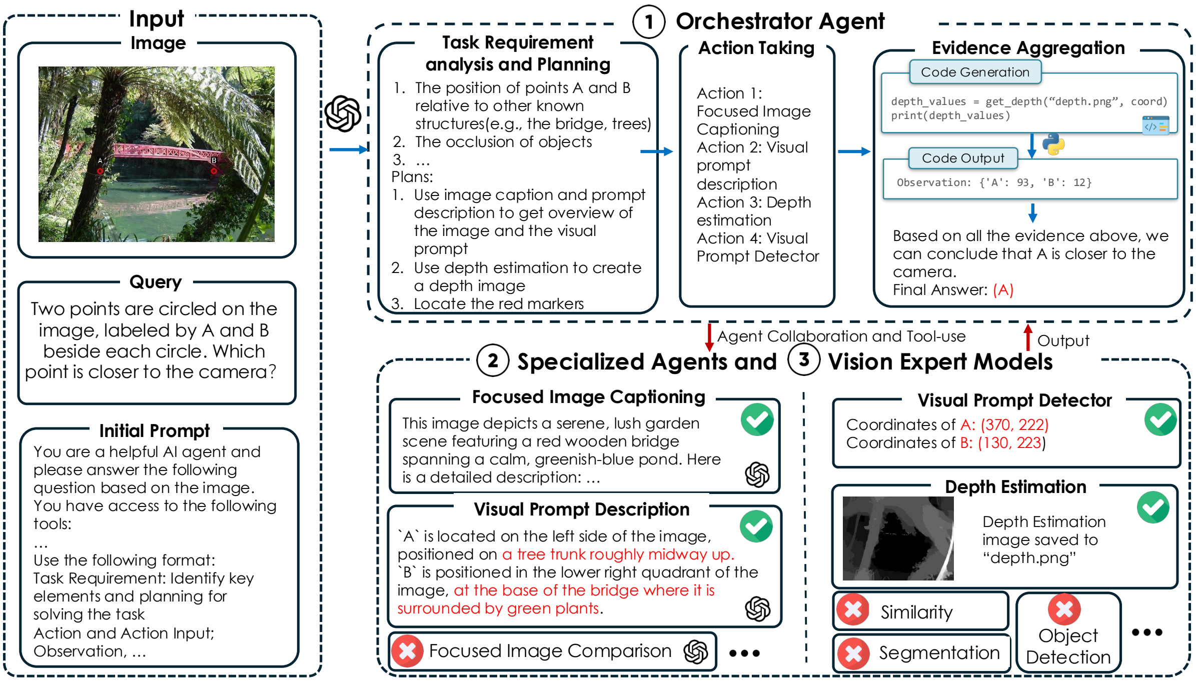
3.1 Orchestrator Agent

Task Requirement Analysis and Planning: Inspired by recent works (Yao et al., 2022; Huang et al., 2022; Yang et al., 2023c; Wang et al., 2023; Sun et al., 2024) that integrate reasoning, planning, and action in LLM-based agent frameworks, the orchestrator agent begins by analyzing the task requirements derived from the images and queries. This analysis identifies the key elements necessary to solve the problem and the corresponding critical visual features that must be acquired in subsequent steps of the agent's workflow, as well as other criteria derived from its own knowledge. The orchestrator agent then generates a detailed plan for tackling the task, outlining the concrete steps required to obtain the information needed to meet these requirements. For instance, in a depth estimation task as illustrated in Figure 1, the orchestrator agent would determine the essential requirements for comparing depth, such as identifying the specific objects targeted by the red circles and recognizing their relative positions to the camera.

Tool Selection and Incorporation of Specialized Agents: After analyzing the task requirements and formulating a plan, the orchestrator agent selects the appropriate tools and specialized agents to provide the visual information necessary to solve the task. Depending on the nature of the task, this may involve initiating collaboration with specialized agents or external vision expert models to gather fine-grained information. Details on these specialized agents and external vision expert models are provided in Sections 3.2 and 3.3.

Evidence Summarization: Once the tools and specialized agents have performed their respective tasks in separate environments, the orchestrator agent compiles and summarizes the collected evidence. This involves integrating the outputs from various tools and agents, ensuring that all relevant information is coherently synthesized to support the decision-making process. The orchestrator agent also resolves conflicting evidence and double-checks the factuality of the information, as errors or hallucinations may arise from the expert models and specialized agents.

Final Answer Deduction: With the summarized evidence, the orchestrator agent deduces the final answer. It applies reasoning based on the accumulated information to arrive at an unambiguous conclusion. Depending on the nature and format of the gathered data, the orchestrator agent may generate `Python` code, which is then executed by an external `Python` interpreter to derive the final answer. If the gathered information does not lead to a perfect answer, the orchestrator agent is designed to select the closest possible option based on the evidence, supplemented by its own understanding.



Two points are circled on the image, labeled by A and B beside each circle. Which point is closer to the camera?

You are a helpful AI agent and please answer the following question based on the image. You have access to the following tools:
...
Use the following format:
Task Requirement: Identify key elements and planning for solving the task
Action and Action Input;
Observation, ...

1. The position of points A and B relative to other known structures(e.g., the bridge, trees)
2. The occlusion of objects
3. ...

Plans:
1. Use image caption and prompt description to get overview of the image and the visual prompt
2. Use depth estimation to create a depth image
3. Locate the red markers

Action 1: Focused Image Captioning
Action 2: Visual prompt description
Action 3: Depth estimation
Action 4: Visual Prompt Detector

```
depth_values = get_depth("depth.png", coord)
print(depth_values)
```

```
Observation: {'A': 93, 'B': 12}
```

Based on all the evidence above, we can conclude that A is closer to the camera.
Final Answer: (A)

This image depicts a serene, lush garden scene featuring a red wooden bridge spanning a calm, greenish-blue pond. Here is a detailed description: ...

`A` is located on the left side of the image, positioned on a tree trunk roughly midway up. `B` is positioned in the lower right quadrant of the image, at the base of the bridge where it is surrounded by green plants.

Coordinates of A: (370, 222)
Coordinates of B: (130, 223)

Depth Estimation image saved to "depth.png"

Figure 1: The VIPACT framework for visual perception. It consists of (1) an orchestrator agent for task analysis and coordination, (2) specialized agents for specialized visual analysis, and (3) vision expert models for providing pixel-level visual information. Note that not all agents and expert models are invoked in every instance—the orchestrator agent selectively activates the most relevant components based on the task characteristics and data. For complete task-solving processes of VIPACT, refer to the case studies in Appendix D.

3.2 Collaboration with Specialized Agents

VIPACT incorporates three specialized agents to enhance its visual perception capabilities: focused image captioning, visual prompt description, and focused image comparison. These agents provide task-specific, detailed information to the orchestrator agent through function calling in a separate environment, integrating their outputs into the main reasoning process. The three specialized agents used in our experiments are described below.

Focused Image Captioning: This agent generates detailed image descriptions, optionally emphasizing specific elements relevant to the task by specifying a `focus` argument. The `focus` argument allows for targeted analysis, ranging from general descriptions to particular aspects like "a red car and the background buildings." This flexibility enables the orchestrator agent to obtain precise, task-relevant information from images. Empirical evidence demonstrates its effectiveness across various tasks, with the focus parameter providing fine-grained control over the generated descriptions.

Visual Prompt Description: Specializing in analyzing visual prompts within images (e.g., colored circles, bounding boxes, arrows, textual labels), this agent is crucial for interpreting visual annotations. It generates detailed descriptions of these elements, including their locations, characteristics, and most importantly, the regions or objects these visual prompts target. This enables the orchestrator agent to accurately interpret highlighted or annotated image sections. The agent has shown particular efficacy in tasks involving images with explicit visual prompts, significantly enhancing the system's ability to understand and reason about annotated visual data.

Focused Image Comparison: This agent analyzes multiple images, identifying similarities and differences with an optional focus on specific elements. Similarly, the `focus` parameter allows for targeted comparative analysis, either generally or on specific features as directed by the orchestrator agent. This function can provide a detailed comparison of orientations of objects which can be useful in tasks such as multi-view reasoning. This capability is valuable for tasks requiring multi-image input, such as change detection or pattern identification across images. Empirical results demonstrate this agent's exceptional effectiveness in tasks involving multiple image inputs, with the focus parameter enabling precise comparative analyses.

The prompts for these specialized agents are in

Appendix J. VIPACT decomposes complex visual tasks into sub-tasks handled by specialized agents, with an orchestrator agent integrating their outputs. The architecture is extensible, allowing for the addition of new agents to address emerging tasks.

3.3 Integration of Vision-Expert Models

VIPACT further enhances its visual perception capabilities by integrating a suite of vision-expert models, each specializing in specific aspects of image analysis. These models collaborate with the orchestrator agent through function calling, uniquely returning both textual data and processed images—making VIPACT among the earliest agent frameworks that incorporate **visual information directly into the reasoning workflow**. These vision-expert models provide fine-grained visual perception information that is often lacking in current VLM's pre-training data (Zhang et al., 2024c). The vision expert tools used in our experiments are described below: (1) **Visual Prompt Detector:** Identifies and localizes annotated elements in images, such as circles, bounding boxes, or other highlighted regions. This tool is crucial for understanding visual instructions or annotations, enabling the agent to focus on relevant areas for analysis. It returns the coordinates of these visual prompts, which often serve as intermediate information to achieve the final answer. (2) **Depth Estimator:** Analyzes spatial relationships within scenes, providing crucial information about the relative distances of objects from the camera. This tool enhances the agent's understanding of 3D structure in 2D images, vital for spatial reasoning tasks. It returns a greyscale depth image that can be directly input into the orchestrator agent, allowing it to interpret depth information or combine it with other evidence to reach the final answer. (3) **Object Detection:** Identifies and localizes objects within an image, providing the agent with a comprehensive inventory of visible objects, their locations, and sizes. This facilitates detailed scene understanding and object-centric reasoning. The tool returns both a processed image with detected objects' bounding boxes and textual information about these bounding boxes and objects. (4) **Image Segmentation:** Offers precise delineation of image regions, separating objects, backgrounds, and distinct areas. This enables fine-grained analysis of image components, crucial for tasks requiring detailed understanding of object boundaries and spatial relationships. It returns images with segmentation masks along with corresponding textual information. (5) **Embedding-based Similarity Computation:** Quantifies visual similarities across images or image regions by generating compact representations of visual content. This allows for nuanced comparisons and similarity assessments, particularly useful for tasks involving image retrieval or comparative analysis. It returns similarity scores based on the selected embedding model and specified similarity metrics, such as cosine similarity. The complete function heads, including inputs, outputs, and descriptions for these vision expert models, are provided in the initial prompt for the orchestrator agents in Appendix J. This diverse toolkit empowers the orchestrator agent to dynamically select and deploy the most appropriate tools for each task, significantly enhancing the framework's ability to comprehend and reason about complex visual scenarios. The integration of processed images alongside textual outputs in the agent's workflow enables more nuanced and contextually rich visual reasoning. We provide an overview of the VipAct framework in Algorithm 1 with detailed explanations in Appendix G.

4 Experiment

Setup. We use various SOTA closed-source models, including **GPT-4o** (OpenAI, 2024), **Gemini-1.5-Pro** (Team et al., 2024), and **Claude-3.5-Sonnet** (Anthropic, 2024), as well as open-source VLMs, such as **LLaVA-OneVision-7B** (Li et al., 2024a), **InternVL-2-Pro** (Chen et al., 2023d, 2024a), and **Llama-3.2-90b-Vision** (Dubey et al., 2024). Following prior works (Zheng et al., 2024a; He et al., 2024a; Liu et al., 2024; Gu et al., 2024), we focus on GPT-4o (OpenAI, 2024) as the primary VLM for analysis in the main paper due to page constraints. Discussions on other VLMs are included in Appendix C, with additional implementation details provided in Appendix A.

Datasets. To evaluate VLMs on visual perception tasks, we use the following two challenging datasets designed to test fine-grained visual perception: (1) **Blink** (Fu et al., 2024) includes diverse visual tasks solvable by humans "within a blink," yet difficult for SOTA VLMs. It features visual prompts such as bounding boxes and interleaved image-text formats, often with multiple images in a single query. We use Blink as the main benchmark. (2) **MMVP** (Tong et al., 2024) is a benchmark for evaluating visual grounding in VLMs, using image pairs from "CLIP-blind pairs"—visually distinct images that are similar in CLIP embedding space. It focuses on nine basic visual patterns that are easy for humans but challenging for SOTA VLMs. Dataset details are in Appendix B.

Baselines. We evaluate VIPACT against four types of baselines: (1) **Text-based prompting**, including zero-shot instructional prompting; chain-of-thought (CoT) prompting (Wei et al., 2022; Kojima et al., 2022); Least-to-most prompting (LtM) (Zhou et al., 2022); and Tree-of-thought (ToT) prompting (Yao et al., 2024). (2) **Few-shot in-context learning** (Brown, 2020), where in-context exemplars are selected using different strategies, including random selection, or selection based on embedding (Radford et al., 2021; Dosovitskiy et al., 2020) similarity (analyzed separately in Appendix E). (3) **Visual Prompting**, exemplified by Set-of-Mark (SoM) (Yang et al., 2023a), which overlays interpretable marks on semantically meaningful image regions. (4) **Vision language agentic frameworks**, including MM-ReAct (Yang et al., 2023c), which integrates LLMs with vision experts via ReAct-style prompts (Yao et al., 2022); ViperGPT (Surís et al., 2023), using LLMs to generate code composing vision and language models; and VisProg (Gupta and Kembhavi, 2023), which generates visual programs from textual instructions.

Result Analysis. Tables 2 and 3 present the performance of our proposed VIPACT framework and baseline methods on each sub-task of the Blink and MMVP datasets respectively. We make the following key observations: **(1) Text-based prompting methods do not consistently improve performance over zero-shot prompting.** Specifically, as shown in Tables 2 and 3, prior text-based prompting methods effective for LLMs — such as CoT — can improve performance on some sub-tasks like visual similarity, object localization, counting, and spatial relations. However, for other tasks, the improvement is minimal or even negative. More advanced techniques like LtM and ToT exhibit similar phenomena. Empirically, while these methods elicit detailed reasoning, such steps are often ungrounded in visual elements and can cause severe hallucinations. Therefore, it is non-trivial to elicit VLMs' reasoning for better general visual perception using text-based methods from text-only LLMs. **(2) SoM can impair VLMs' fine-grained perception in most scenarios.** From results on both datasets, SoM adversely affects VLM performance on almost all tasks. Empirically, overlaying labeled masks can become cluttered with numerous semantic objects or fine-grained parts, negatively influencing VLM perception of original objects and potentially confusing models with original visual prompts and labels. Consequently, SoM's effectiveness in some compositional reasoning tasks with limited semantic objects does not generalize well to broader visual perception tasks, especially those requiring visual prompt understanding. **(3) Previous visual programming methods exhibit poor generalization ability.** As shown, these methods perform adequately only on limited tasks (e.g., spatial relations, counting) similar to those in common VQA datasets (Hudson and Manning, 2019; Suhr et al., 2019; Marino et al., 2019). Their generated code calls a limited set of predefined tools, lacking logic for unsupported scenarios or errors. They cannot support images with visual prompts, failing to locate them for subsequent operations (e.g., near-zero performance in depth estimation due to inability to locate red circles, leading to non-executable code). Moreover, code generated solely from text queries lacks flexibility for different image characteristics. These observations highlight the need for a generalizable agent framework leveraging both vision expert models and VLM flexibility. **(4) VIPACT consistently achieves the best performance across all sub-tasks in Blink and MMVP, demonstrating its effectiveness and generalization ability.** By examining VIPACT's reasoning traces, we observe that, compared to text-based and visual prompting methods, VIPACT effectively invokes specialized agents or vision expert models to enhance image understanding. It does not solely rely on their outputs, as evidence might be incorrect or errors may occur. Instead, it aggregates useful evidence with additional reasoning to infer the final answer, showcasing its ability to handle uncertainties and integrate multiple information sources. Figure 3 and 4 in Appendix D show complete reasoning traces of VIPACT.

5 Ablation Study

To evaluate the effectiveness of various components in our VIPACT framework, we further conduct a series of ablation studies. These studies involve removing or modifying key components of the VIPACT framework to assess their impact on performance across different visual reasoning tasks. The ablation studies are as follows: **(1) Removal of multi-agent collaboration**: We removed the specialized agents and incorporated their prompts

Method	Sim	Count	Depth	Jig	Fun.C	Sem.C	Spat	Local	Vis.C	Multi-v	Average
Text-based Prompting w/ GPT-4o											
Zero-shot	65.44	50.83	64.52	60.00	57.69	56.83	79.92	56.00	86.05	60.15	63.74
CoT	63.70	65.00	73.39	62.00	57.69	57.55	82.52	60.66	82.56	53.38	65.85
LtM	62.22	64.17	70.97	62.67	55.38	55.40	76.22	59.02	83.14	45.86	63.51
ToT	64.44	58.33	71.70	64.00	57.69	59.71	83.22	61.48	78.49	50.38	64.94
Visual Prompting w/ GPT-4o											
SoM	63.70	43.33	68.55	49.33	47.69	52.52	76.22	59.84	83.72	56.40	60.13
Mutli-modal Agent Framework w/ GPT-4o											
MM-ReAcT	-	30.00	0.81	-	-	-	63.64	0.00	-	-	-
ViperGPT	-	29.17	0.00	-	-	-	48.95	18.85	-	-	-
VisProg	-	3.33	0.00	-	-	-	31.47	14.75	-	-	-
VipAct (Ours)	**81.48**	**70.00**	**90.80**	**68.00**	**61.50**	**60.40**	**86.70**	**63.11**	**91.28**	**62.63**	**73.79**

Table 2: Results for visual reasoning tasks in Blink using GPT-4o. Note that "–" indicates methods that do not support multiple images. Our VipAct consistently outperforms baselines on all tasks.

Method	Accuracy (%)
Zero-shot	68.0
CoT	61.0
LtM	66.0
ToT	66.0
SoM	62.0
MM-ReAct	6.67
ViperGPT	53.0
VisPro	39.0
VipAct (Ours)	**70.7**

Table 3: Different methods using GPT-4o on MMVP.

as instructions directly into the orchestrator agent to evaluate the importance of multi-agent collaboration. **(2) Removal of image input for orchestrator agent:** We modified the input to the orchestrator agent to only include image paths as text, rather than the actual images which means the image is not visible to the orchestrator agent but still can be served as input for other specialized agents or vision expert models. This setup follows the paradigm used in previous works (Surís et al., 2023; Gupta and Kembhavi, 2023) and tests the effectiveness of direct visual input to the orchestrator agent. **(3) Removal of specialized agents:** We removed all specialized agents to assess their impact on the VipAct's performance. **(4) Removal of vision expert models:** We eliminated all vision-expert models to evaluate their contribution.

The results of these ablation studies are presented in Table 4 and 5. From these results, we derive the following key insights: **(1) Multi-agent collaboration enhances detailed reasoning**: The removal of multi-agent collaboration led to a consistent performance decline. By comparing reasoning steps, we observed that multi-agent collaboration enabled significantly more detailed image analysis (over 80% more generated tokens), such as thorough image captioning. This aligns with observations in LLMs (Wu et al., 2023b; Hong et al., 2023; Qian et al., 2023; Park et al., 2023; Liu et al., 2023b), where agent collaboration enhances complex task-solving via comprehensive reasoning from diverse perspectives. **(2) Direct image input to the orchestrator agent is essential for flexible task planning and error handling**: As shown in Tables 4 and 5, removing direct image input to the orchestrator significantly degrades performance. Without direct visual access, the orchestrator agent relies solely on textual queries, lacking critical visual information for accurate task planning. This leads to suboptimal decision-making, less precise parameter selection (e.g., the `focus` parameter), and overly generalized task analysis, reducing specificity. For instance, in a multi-view reasoning task, direct image input allows the agent to identify reference objects, enabling it to accurately adjust the `focus` parameter and effectively determine the direction of camera movement. Further analysis is in Appendix H. **(3) Specialized agents and vision expert models significantly contribute to performance**: Specialized agents, though VLMs, intently analyze specific visual information without distractions from other instructions (e.g., format requirements), which can hinder LLM reasoning (Tam et al., 2024). Vision expert models perform pixel-level analyses beyond SOTA VLM capabilities, aiding the orchestrator. As demonstrated in Table 4 and 5, removing these components leads to a noticeable performance decline. Overall, our VipAct framework combines

Method	Sim	Count	Depth	Jig	Fun.C	Sem.C	Spat	Local	Vis.C	Multi-v
Variants of VipAct										
VipAct (Full)	**81.48**	**70.00**	**90.80**	**68.00**	**61.50**	**60.40**	**86.70**	63.11	**91.28**	**62.63**
w/o Multi-agent	80.00	67.50	75.00	66.00	58.46	59.71	82.52	**63.93**	85.47	48.87
w/o Visual Input	77.78	59.71	69.35	61.33	53.85	51.08	83.22	60.66	78.49	48.12
w/o Spec. Agents	65.72	62.45	85.62	62.32	55.25	56.32	81.96	58.49	75.48	46.75
w/o Vision Expert	64.34	57.44	72.58	65.67	59.42	58.59	81.37	57.44	83.63	56.40

Table 4: Ablation study results of VipAct on the Blink benchmark using GPT-4o. VipAct (Full) represents the complete framework with all components, while the other variants exclude specific components.

Method	Accuracy (%)
VipAct	**70.7**
w/o Multi-agent	68.0
w/o Visual Input	54.0
w/o Spec. Agents	67.0
w/o Vision Expert	66.0

Table 5: Ablation of VipAct on MMVP using GPT-4o.

VLM flexibility and planning with vision expert model precision, creating a cohesive system where each component is essential.

6 Error Analysis

To examine the limitations of GPT-4o's visual perception capabilities as well as the bottlenecks of our VipAct, we conduct a detailed error analysis. Following prior works (Zhou et al., 2022; Chen et al., 2023b; Zhang et al., 2024d), we randomly sampled 20 error cases from each sub-task within the two datasets. The errors were categorized as follows: (1) **Failure to perceive small object parts (17%):** The model often overlooks small, semantically important components crucial for precise visual understanding. (2) **Difficulty distinguishing closely positioned visual prompts (15%):** The model struggles to differentiate spatially proximate visual prompts. (3) **Challenges in fine-grained spatial reasoning (24%):** Tasks requiring high spatial resolution highlight the model's **bias towards foreground objects over backgrounds**; e.g., misinterpreting a highlight meant for the sky near a car as associated with the car. (4) **Misinterpretation of relative object positions (14%):** Errors arise when object arrangements differ from real-world expectations, as the model often lacks the ability to infer spatial relations from objects' perspectives, focusing on camera viewpoint. (5) **Failure to recognize object orientation (13%):** Difficulty discerning object orientation leads to errors in recognizing object parts, such as distinguishing left/right bicycle pedals. (6) **Other errors (17%):** This includes other issues like failure to detect subtle color differences, inaccuracies in multi-image fine-grained structure correspondence, and instances of refusal or instruction misinterpretation.

Case studies illustrating these errors are in Appendix D. Our analysis denotes that while VipAct shows significant improvements in VLM visual perception, fine-grained perception remains a bottleneck. Specifically, the model lacks the **spatial intelligence or imaginative abilities** (Chen et al., 2018; Huang et al., 2024) necessary to infer relative object positions beyond their pixel positions (from the camera's perspective), especially in the context of real-life scenes. Noticeably, these limitations hinder the model's ability to accurately interpret visual prompts and process tasks involving multiple image inputs. We also examine the significance of multiple image inputs for VLMs in Appendix F.

7 Conclusion

We introduce **VipAct**, a VLM-based agent framework that synergizes multi-agent collaboration and vision expert models for fine-grained visual perception tasks. By combining the planning and function-calling capabilities of SOTA VLMs, VipAct enhances VLMs' System-2 reasoning through multi-agent interactions and integrates high-precision, pixel-level information from specialized vision models. Our experiments across a diverse range of visual perception tasks demonstrate that VipAct achieves SOTA performance, outperforming previous baselines. The comprehensive ablation study highlights the critical role of multi-agent collaboration in eliciting detailed information for reasoning, as well as the importance of image input in task planning. Furthermore, our error analysis highlights several inherent limitations in current SOTA VLMs that form bottlenecks in our framework, offering valuable insights for future improvements.

Limitations

Our work has several limitations: (1) The inference cost of VLMs is high, as VIPACT requires multiple inferences, leading to increased computational overhead. This is a common issue across all multi-agent frameworks that involve complex reasoning steps, and it is inevitable when generating more detailed reasoning. (2) VIPACT relies on closed-source VLMs for their superior instruction-following and function-calling capabilities. While other open-source alternatives are explored in Appendix C, they struggle with following instructions such as formatting requirements. However, VIPACT is a general framework and can be adapted to other VLMs as they evolve. (3) Task-specific vision expert tools were not developed for all scenarios; however, the modular architecture allows seamless integration of additional tools and agents.

References

Sweta Agrawal, Chunting Zhou, Mike Lewis, Luke Zettlemoyer, and Marjan Ghazvininejad. 2023. In-context examples selection for machine translation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8857–8873, Toronto, Canada. Association for Computational Linguistics.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, and 1 others. 2022. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736.

Dosovitskiy Alexey. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv: 2010.11929*.

Anthropic. 2024. Claude 3.5 sonnet.

Anas Awadalla, Irena Gao, Josh Gardner, Jack Hessel, Yusuf Hanafy, Wanrong Zhu, Kalyani Marathe, Yonatan Bitton, Samir Gadre, Shiori Sagawa, and 1 others. 2023. Openflamingo: An open-source framework for training large autoregressive vision-language models. *arXiv preprint arXiv:2308.01390*.

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

Ankan Bansal, Yuting Zhang, and Rama Chellappa. 2020. Visual question answering on image sets. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXI 16*, pages 51–67. Springer.

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in NLP. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 995–1005, Jeju Island, Korea. Association for Computational Linguistics.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F Karlsson, Jie Fu, and Yemin Shi. 2023a. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*.

Jiaao Chen, Xiaoman Pan, Dian Yu, Kaiqiang Song, Xiaoyang Wang, Dong Yu, and Jianshu Chen. 2023b. Skills-in-context prompting: Unlocking compositionality in large language models. *arXiv preprint arXiv:2308.00304*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Shuo Chen, Zhen Han, Bailan He, Mark Buckley, Philip Torr, Volker Tresp, and Jindong Gu. 2023c. Understanding and improving in-context learning on vision-language models. *arXiv preprint arXiv:2311.18021*, 1(2).

Xinlei Chen, Li-Jia Li, Li Fei-Fei, and Abhinav Gupta. 2018. Iterative visual reasoning beyond convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi Hu, Jiapeng Luo, Zheng Ma, and 1 others. 2024a. How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites. *arXiv preprint arXiv:2404.16821*.

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, Bin Li, Ping Luo, Tong Lu, Yu Qiao, and Jifeng Dai. 2023d. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. *arXiv preprint arXiv:2312.14238*.

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, and 1 others. 2024b. Internvl:

Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24185–24198.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024a. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36.

Yang Deng, Xuan Zhang, Wenxuan Zhang, Yifei Yuan, See-Kiong Ng, and Tat-Seng Chua. 2024b. On the multi-turn instruction following for conversational web agents. *arXiv preprint arXiv:2402.15057*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and 1 others. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Yue Fan, Lei Ding, Ching-Chen Kuo, Shan Jiang, Yang Zhao, Xinze Guan, Jie Yang, Yi Zhang, and Xin Eric Wang. 2024. Read anywhere pointed: Layout-aware gui screen reading with tree-of-lens grounding. *arXiv preprint arXiv:2406.19263*.

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A Smith, Wei-Chiu Ma, and Ranjay Krishna. 2024. Blink: Multimodal large language models can see but not perceive. *arXiv preprint arXiv:2404.12390*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Yu Gu, Boyuan Zheng, Boyu Gou, Kai Zhang, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. 2024. Is your llm secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*.

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

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024a. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890, Bangkok, Thailand. Association for Computational Linguistics.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024b. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, and 1 others. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. 2024a. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *arXiv preprint arXiv:2406.09403*.

Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. 2024b. 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 (CVPR)*, pages 9590–9601.

Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. 2024c. 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.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. 2024. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*.

Drew A Hudson and Christopher D Manning. 2019. 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.

Muhammad Hussain. 2023. Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, 11(7):677.

Yixing Jiang, Jeremy Irvin, Ji Hun Wang, Muhammad Ahmed Chaudhry, Jonathan H Chen, and Andrew Y Ng. 2024. Many-shot in-context learning in multimodal foundation models. *arXiv preprint arXiv:2405.09798*.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. 2024. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. *arXiv preprint arXiv:2402.17553*.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, and 1 others. 2023. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*.

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

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

Xuanyu Lei, Zonghan Yang, Xinrui Chen, Peng Li, and Yang Liu. 2024. Scaffolding coordinates to promote vision-language coordination in large multi-modal models. *arXiv preprint arXiv:2402.12058*.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. 2024a. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*.

Feng Li, Renrui Zhang, Hao Zhang, Yuanhan Zhang, Bo Li, Wei Li, Zejun Ma, and Chunyuan Li. 2024b. Llava-next-interleave: Tackling multi-image, video, and 3d in large multimodal models. *arXiv preprint arXiv:2407.07895*.

Xiaotong Li, Fan Zhang, Haiwen Diao, Yueze Wang, Xinlong Wang, and Ling-Yu Duan. 2024c. Densefusion-1m: Merging vision experts for comprehensive multimodal perception. *arXiv preprint arXiv:2407.08303*.

Junpeng Liu, Yifan Song, Bill Yuchen Lin, Wai Lam, Graham Neubig, Yuanzhi Li, and Xiang Yue. 2024. Visualwebbench: How far have multimodal llms evolved in web page understanding and grounding? *arXiv preprint arXiv:2404.05955*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2023a. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2023b. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2023. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. 2024. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*.

Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. 2024. A survey on vision-language-action models for embodied ai. *arXiv preprint arXiv:2405.14093*.

Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. 2019. 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.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, and 1 others. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, and 1 others. 2024. Pivot: Iterative visual prompting elicits actionable knowledge for vlms. *arXiv preprint arXiv:2402.07872*.

Tai Nguyen and Eric Wong. 2023. In-context example selection with influences. *arXiv preprint arXiv:2302.11042*.

OpenAI. 2024. Hello gpt-4o. `https://openai.com/index/hello-gpt-4o/`. Accessed: 2024-08-22.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA. Association for Computing Machinery.

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 6.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, and 1 others. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Pooyan Rahmanzadehgervi, Logan Bolton, Mohammad Reza Taesiri, and Anh Totti Nguyen. 2024. Vision language models are blind. *arXiv preprint arXiv:2407.06581*.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, and 1 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Swarnadeep Saha, Omer Levy, Asli Celikyilmaz, Mohit Bansal, Jason Weston, and Xian Li. 2023. Branch-solve-merge improves large language model evaluation and generation. *arXiv preprint arXiv:2310.15123*.

Swarnadeep Saha, Archiki Prasad, Justin Chih-Yao Chen, Peter Hase, Elias Stengel-Eskin, and Mohit Bansal. 2024. System-1. x: Learning to balance fast and slow planning with language models. *arXiv preprint arXiv:2407.14414*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

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

Lin Shi, Weicheng Ma, and Soroush Vosoughi. 2024a. Judging the judges: A systematic investigation of position bias in pairwise comparative assessments by llms. *arXiv preprint arXiv:2406.07791*.

Min Shi, Fuxiao Liu, Shihao Wang, Shijia Liao, Subhashree Radhakrishnan, De-An Huang, Hongxu Yin, Karan Sapra, Yaser Yacoob, Humphrey Shi, and 1 others. 2024b. Eagle: Exploring the design space for multimodal llms with mixture of encoders. *arXiv preprint arXiv:2408.15998*.

Significant-Gravitas. 2024. Autogpt. GitHub repository.

Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, Kevin Yang, Arsha Nagrani, Cordelia Schmid, Andy Zeng, Trevor Darrell, and Dan Klein. 2023. Modular visual question answering via code generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 747–761, Toronto, Canada. Association for Computational Linguistics.

Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. 2019. A corpus for reasoning about natural language grounded in photographs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6418–6428, Florence, Italy. Association for Computational Linguistics.

Simeng Sun, Yang Liu, Shuohang Wang, Dan Iter, Chenguang Zhu, and Mohit Iyyer. 2024. PEARL: Prompting large language models to plan and execute actions over long documents. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 469–486, St. Julian's, Malta. Association for Computational Linguistics.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11888–11898.

Andrew Szot, Bogdan Mazoure, Harsh Agrawal, Devon Hjelm, Zsolt Kira, and Alexander Toshev. 2024. Grounding multimodal large language models in actions. *arXiv preprint arXiv:2406.07904*.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. 2024. Let me speak freely? a study on the impact of format restrictions on performance of large language models. *arXiv preprint arXiv:2408.02442*.

Weihao Tan, Ziluo Ding, Wentao Zhang, Boyu Li, Bohan Zhou, Junpeng Yue, Haochong Xia, Jiechuan Jiang, Longtao Zheng, Xinrun Xu, and 1 others. 2024. Towards general computer control: A multimodal agent for red dead redemption ii as a case study. *arXiv preprint arXiv:2403.03186*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, and 1 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Shengbang Tong, Zhuang Liu, Yuexiang Zhai, Yi Ma, Yann LeCun, and Saining Xie. 2024. Eyes wide shut? exploring the visual shortcomings of multimodal llms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9568–9578.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2024a. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024b. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024c. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, and 1 others. 2024d. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023b. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Yixuan Wu, Yizhou Wang, Shixiang Tang, Wenhao Wu, Tong He, Wanli Ouyang, Jian Wu, and Philip Torr. 2024. Dettoolchain: A new prompting paradigm to unleash detection ability of mllm. *arXiv preprint arXiv:2403.12488*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024a. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Junlin Xie, Zhihong Chen, Ruifei Zhang, Xiang Wan, and Guanbin Li. 2024b. Large multimodal agents: A survey. *arXiv preprint arXiv:2402.15116*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024c. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*.

An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, and 1 others. 2023. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023a. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*.

Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. 2024a. Depth anything v2. *arXiv preprint arXiv:2406.09414*.

Yijun Yang, Tianyi Zhou, Kanxue Li, Dapeng Tao, Lusong Li, Li Shen, Xiaodong He, Jing Jiang, and Yuhui Shi. 2024b. 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 (CVPR)*, pages 26275–26285.

Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023b. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024. Ferret-ui: Grounded mobile ui understanding with multimodal llms. *arXiv preprint arXiv:2404.05719*.

Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. 2024. Distilling system 2 into system 1. *arXiv preprint arXiv:2407.06023*.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, and 1 others. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9556–9567.

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, and 1 others. 2024a. Ufo: A ui-focused agent for windows os interaction. *arXiv preprint arXiv:2402.07939*.

Jintian Zhang, Xin Xu, and Shumin Deng. 2023a. Exploring collaboration mechanisms for llm agents: A social psychology view. *arXiv preprint arXiv:2310.02124*.

Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. 2024b. Android in the zoo: Chain-of-action-thought for gui agents. *arXiv preprint arXiv:2403.02713*.

Ruiyi Zhang, Yufan Zhou, Jian Chen, Jiuxiang Gu, Changyou Chen, and Tong Sun. 2024c. Llava-read: Enhancing reading ability of multimodal language models. *arXiv preprint arXiv:2407.19185*.

Yiming Zhang, Shi Feng, and Chenhao Tan. 2022. Active example selection for in-context learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9134–9148, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yuanhan Zhang, Kaiyang Zhou, and Ziwei Liu. 2023b. What makes good examples for visual in-context learning? *Advances in Neural Information Processing Systems*, 36:17773–17794.

Zhehao Zhang, Jiaao Chen, and Diyi Yang. 2024d. Darg: Dynamic evaluation of large language models via adaptive reasoning graph. *arXiv preprint arXiv:2406.17271*.

Zhehao Zhang, Yan Gao, and Jian-Guang Lou. 2024e. e^5: Zero-shot hierarchical table analysis using augmented LLMs via explain, extract, execute, exhibit and extrapolate. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1244–1258, Mexico City, Mexico. Association for Computational Linguistics.

Zhehao Zhang, Xitao Li, Yan Gao, and Jian-Guang Lou. 2023c. CRT-QA: A dataset of complex reasoning question answering over tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2131–2153, Singapore. Association for Computational Linguistics.

Zheng Zhang, Fan Yang, Ziyan Jiang, Zheng Chen, Zhengyang Zhao, Chengyuan Ma, Liang Zhao, and Yang Liu. 2024f. Position-aware parameter efficient fine-tuning approach for reducing positional bias in llms. *arXiv preprint arXiv:2404.01430*.

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

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024a. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.

Longtao Zheng, Zhiyuan Huang, Zhenghai Xue, Xinrun Wang, Bo An, and Shuicheng Yan. 2024b. Agentstudio: A toolkit for building general virtual agents. *arXiv preprint arXiv:2403.17918*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and 1 others. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A Implementation Details

For main experiments, we use the `gpt-4o-2024-05-13` model from Azure OpenAI API. Following previous works (Fu et al., 2024) to ensure reproducibility, we set the temperature to 0 for all VLM inference and set the maximum number of tokens to 2048. For components of VIPACT, we use the same `gpt-4o-2024-05-13` model for the implementation of orchestrator agents and specialized agents. For the implementation of vision expert models, we use the `Depth-Anything-V2-Small-hf` checkpoint (Yang et al., 2024a) for depth estimation, the Segment Anything Model (SAM) (Kirillov et al., 2023) for segmentation, the YOLOv8 model (Hussain, 2023) from Ultralytics for object detection, and the `clip-vit-base-patch32` (Radford et al., 2021) for similarity comparison using cosine similarity. For experiments with LLaVA, we use the latest SOTA `llava-onevision-qwen2-7b-ov` (Li et al., 2024a), which is one of the few VLMs that support multiple images as inputs and achieves SOTA results on various vision-language benchmarks (Li et al., 2024b; Bansal et al., 2020) compared to other open-source models of similar size. For the implementation of all prompting baselines, we adopt the codebase from the original Blink and MMVP papers and use the exact same settings, including the method for computing performance. For the implementation of baselines MM-ReAct, ViperGPT, and VisProg, we adopt the original codebase they provide, except that the backbone model is replaced with GPT-4o, as their original models such as Codex (Chen et al., 2021) are not available and to ensure fair comparison. For the implementation of few-shot in-context learning, the embedding models' checkpoints we use are `clip-vit-base-patch32` and `vit-base-patch16-224` (Alexey, 2020). For all experiments, we run three times and report the average number. For the results in Table 2 and 3, we conduct significance tests following (Berg-Kirkpatrick et al., 2012). The average estimate of p-value is 0.006 (< 0.01) between VIPACT and SOTA baselines, demonstrating significant differences. The total inference time for our VIPACT on Blink and MMVP is less than 2 hours, which is acceptable. Our computational resources consist of a Linux server with 4 NVIDIA A100-40G GPUs.

B Dataset Details

In this section, we provide the details of the dataset used in our experiments. The Blink (Fu et al., 2024) dataset contains a variety of tasks that evaluate different aspects of VLMs' perception capabilities. In our paper, we specifically focus on the following sub-tasks: Similarity (**Sim**), Counting (**Count**), Depth Estimation (**Depth**), Jigsaw Puzzle (**Jig**), Functional Correspondence (**Fun.C**), Semantic Correspondence (**Sem.C**), Spatial relation (**Spat**), Local Correspondence (**Local**), Visual Correspondence (**Vis.C**), and Multi-view Reasoning (**Multi-v**). The dataset is divided into validation and test sets, with the number of data points for each sub-task as shown in Table 6.

The tasks and the corresponding datasets are described in the original Blink paper. Each sub-task is designed to challenge different aspects of the model's perceptual reasoning capabilities, as detailed in the main text of our paper. Following previous works (Hu et al., 2024c), we exclude datasets focused on compositional reasoning like IQ testing or commonsense reasoning, as they do not directly assess visual perception and more focus on compositional reasoning.

Another dataset we use in this work is the Multimodal Visual Patterns (MMVP) dataset (Tong et al., 2024) which consists of 150 CLIP-blind image pairs and 300 associated visual questions, designed to probe nine core visual patterns: orientation, presence of specific features, state, quantity, positional context, color, structure, text, and viewpoint. Human participants achieved 95.7% accuracy, while state-of-the-art MLLMs, including GPT-4V and Gemini, performed significantly worse. The dataset highlights fundamental failures in visual grounding tasks and serves as a benchmark for advancing VLMs' visual perception ability.

C Exploration of different VLMs

In addition to the GPT-4o used in our main experiments, we also evaluate other VLMs on our tasks. Specifically, we explore five additional SOTA VLMs, including (1) open-source models: LLaVA-OneVision-7B (Li et al., 2024a), the latest open-source model in the LLaVA series, InternVL-2-Pro (Chen et al., 2023d, 2024a), and Llama-3.2-90b-vision (Dubey et al., 2024); and (2) close-source models: Gemini-1.5-Pro (Team et al., 2024) and Claude-3.5-Sonnet (Anthropic, 2024).

For open-source models, we find that applying VIPACT's prompt (described in Section J) reveals significant limitations. These VLMs often fail to follow key instructions, such as adhering to the required format, which is critical for extracting the tool-use indicators necessary for integrating external tools. Furthermore, they frequently generate image captions even when no such instruction is provided, suggesting a bias towards image captioning or description tasks.

Sub-task	**Sim**	**Count**	**Depth**	**Jig**	**Fun.C**	**Sem.C**	**Spat**	**Local**	**Vis.C**	**Multi-v**
Validation	135	120	124	150	130	139	143	122	172	133
Test	136	120	124	150	130	140	143	125	172	133

Table 6: Number of data points for each sub-task in the validation and test sets of Blink.

To evaluate these open-source VLMs comprehensively, we apply prompting baselines and report the results on the Blink benchmark and MMVP in Table 9 and Table 8. These results demonstrate that while LLaVA-OneVision-7B achieves above-random accuracy on tasks like object counting and spatial relations—typical of standard VQA problems found in prior datasets (Li et al., 2024b; Bansal et al., 2020)—it performs near or below random on other tasks. We also observe significant positional biases (Zhang et al., 2024f; Shi et al., 2024a), with this model frequently predicting the first option for most data points within a task. In contrast, InternVL-2-Pro and Llama-3.2-90b-vision exhibit better performance, though still significantly behind GPT-4o. These findings indicate that current open-source SOTA VLMs struggle with generalizing to more complex or non-standard VQA tasks, lacking the fine-grained perception capabilities necessary for broader applicability. Moreover, alternative prompting strategies do not yield noticeable improvements over the zero-shot baseline for these models.

In contrast, the two additional close-source VLMs—Gemini-1.5-Pro and Claude-3.5-Sonnet—demonstrate instruction-following abilities comparable to GPT-4o, allowing effective application of our VIPACT framework. As shown in Tables 7, applying VIPACT on these models consistently outperforms previous prompting baselines, achieving significant improvements. These results highlight the effectiveness and generalization capability of our approach when used with models possessing strong instruction-following capabilities.

D Case Studies

To intuitively demonstrate the effectiveness of our proposed VIPACT and highlight the bottlenecks of current SOTA VLMs, we present a series of case studies showcasing both failure (Figure 2) and success cases (Figures 3 and 4) of our method.

In Figure 2, we observe instances where VLM-based specialized agents in VIPACT make reasoning errors, as categorized in Section 6. Although VIPACT includes an error-handling mechanism to reassess the evidence, these errors can still mislead the orchestrator agent, leading to incorrect inferences. For instance, in the top case of Figure 2, the VLM fails to accurately infer the orientation of the bicycle in the left image, mistakenly identifying the left pedal as the reference point based on the camera's perspective. In the middle case, the VLM overlooks the small portion of the cap's brim, leading to an incorrect prediction. Finally, the bottom case demonstrates how the camera's perspective makes it appear as though the apples are positioned above the orange when in reality, they are on the same plate at the same height. These examples highlight the limitations in visual intelligence exhibited by SOTA VLMs such as GPT-4o, particularly in tasks requiring fine-grained spatial reasoning.

In Figures 3 and 4, we present two examples that demonstrate the complete reasoning process of our VIPACT, integrating vision expert models and specialized agents. Figure 3 illustrates a scenario where the orchestrator agent sequentially invokes vision expert models, including a Visual Prompt Detector and a Depth Estimator, to accurately determine the depth values of two red points in the image, ultimately arriving at the correct answer. In contrast, we observe that GPT-4o is unable to perceive such depth information on its own. Figure 4 presents a case where no existing vision tools can directly solve the problem. Here, the orchestrator agent introduces a specialized agent specifically designed for visual prompt description. This agent provides a detailed analysis of each visual prompt (marked by red circles) in the second image, leading to the correct prediction. These two examples effectively illustrate the strength of our VIPACT framework in integrating vision expert models and specialized agents to enhance reasoning capabilities.

Method	Sim	Count	Depth	Jig	Fun.C	Sem.C	Spat	Local	Vis.C	Multi-v	Overall
Text-based Prompting w/ Gemini-1.5-Pro											
Zero-shot	78.52	60.83	70.97	72.67	44.62	51.08	74.13	57.38	81.98	55.64	64.78
CoT	81.48	**64.17**	78.23	68.67	42.31	53.96	78.32	60.66	81.98	51.88	66.17
LtM	84.44	62.50	75.81	73.33	46.92	50.36	73.43	63.11	84.30	54.14	66.83
ToT	82.23	61.52	74.84	68.96	45.42	52.61	75.34	61.25	82.21	52.86	65.72
Visual Prompting w/ Gemini-1.5-Pro											
SoM	60.74	55.00	65.32	62.00	47.69	43.88	74.13	59.02	74.42	53.38	59.56
Multi-modal Agent Framework w/ Gemini-1.5-Pro											
VipAct	**84.44**	**64.17**	**89.42**	**74.00**	**48.74**	**57.55**	**79.57**	**70.48**	**86.05**	**59.42**	**71.38**
Text-based Prompting w/ Claude-3.5-Sonnet											
Zero-shot	85.19	67.50	66.13	58.00	58.00	44.60	72.03	57.38	73.84	48.12	63.08
CoT	86.72	**68.33**	71.77	61.33	52.31	41.73	77.62	50.00	81.98	44.36	63.62
LtM	87.42	67.42	68.42	59.97	58.00	45.13	73.82	57.47	74.29	47.86	63.98
ToT	86.90	67.53	69.48	57.35	59.46	43.72	74.92	58.49	76.14	46.38	64.04
Visual Prompting w/ Claude-3.5-Sonnet											
SoM	82.65	62.78	63.81	56.79	56.73	39.58	72.00	52.47	73.74	44.63	60.52
Multi-modal Agent Framework w/ Claude-3.5-Sonnet											
VipAct	**88.89**	67.96	**88.59**	**65.33**	**60.42**	**50.13**	**78.82**	**61.54**	**83.72**	**49.57**	**69.50**

Table 7: Results for visual reasoning tasks in Blink using Gemini-1.5-Pro and Claude-3.5-Sonnet. Our `VipAct` consistently outperforms baselines on almost all tasks.

Method	LLaVA-OneVision-7B	InternVL-2-Pro	Llama-3.2-90b-vision
Random	25.00	25.00	25.00
Zero-shot	29.67	60.00	57.33
CoT	30.33	57.33	59.33
LtM	30.00	58.67	57.33
ToT	31.33	60.00	59.38
SoM	27.00	45.33	51.33

Table 8: Results of different open-source VLMs with different prompting methods on the MMVP benchmark, including a random baseline for comparison.

Method	Sim	Count	Depth	Jig	Fun.C	Sem.C	Spat	Local	Vis.C	Multi-v
Random	50.00	25.00	50.00	50.00	25.00	25.00	50.00	50.00	25.00	50.00
Zero-shot	47.41	63.33	51.61	52.67	20.00	23.02	72.73	50.82	23.26	44.36
CoT	44.44	57.20	54.03	52.67	20.77	25.90	76.22	43.44	22.67	35.34
LtM	45.93	56.67	51.61	52.67	15.38	28.87	72.03	50.82	30.81	42.11
ToT	47.41	63.33	50.00	52.67	15.38	24.46	72.03	50.82	23.26	44.36
SoM	47.41	46.67	54.03	52.67	23.85	21.58	72.73	41.80	19.19	31.58

Table 9: Result of baseline methods evaluated using LLaVa-OneVision-7B on the Blink dataset.

E Few-shot In-context Learning

In this section, we examine the effectiveness of few-shot in-context learning in visual perception tasks using various VLMs, including GPT-4o and LLaVA-OneVision-7B. Following previous works (Brown, 2020; Alayrac et al., 2022; Awadalla et al., 2023; Zhao et al., 2023; Jiang et al., 2024), we append a series of (image(s), question, answer) triplets—ranging from 1 to 5—before the test query, within the overall instruction. This setup has been shown to enhance performance in LLMs on a wide range of NLP tasks. Additionally, prior research indicates that LLMs can be sensitive to the selection of in-context exemplars (Nguyen and Wong, 2023;

Dataset	Model	
	GPT-4o	LLaVA-OneVision-7B
Sim	59.51 (↓ -5.93)	45.93 (↓ -1.48)
Jig	57.78 (↓ -2.22)	52.67 (→ 0.00)
Fun.C	53.34 (↓ -4.35)	20.00 (→ 0.00)
Sem.C	56.60 (↓ -0.23)	24.46 (↑ +1.44)
Vis.C	83.91 (↓ -2.14)	18.60 (↓ -4.66)
Multi-v	51.38 (↓ -8.77)	29.32 (↓ -15.04)
Overall	60.42 (↓ -3.94)	31.83 (↓ -3.29)

Table 10: Results of GPT-4o and LLaVA-OneVision-7B on Blink tasks requiring multiple image inputs, where multiple images are concatenated into a single image during inference. Performance changes compared to the zero-shot baseline with multiple image inputs are indicated in parentheses.

Zhang et al., 2022; Agrawal et al., 2023; Chen et al., 2023c; Zhang et al., 2023b). To explore this, we employ three different strategies for exemplar selection: (1) Randomly select a specified number of exemplars. (2) Select exemplars based on top-K similarity using the averaged CLIP embedding of images, which captures both textual semantics and visual information (Radford et al., 2021). (3) Select exemplars based on top-K similarity using ViT embeddings (Alexey, 2020), which focus purely on visual features.

Table 11 presents the results of few-shot in-context learning with GPT-4o on the Blink benchmark. We observe that for certain tasks, such as object counting and spatial relations, few-shot learning significantly decreases performance compared to other baselines (see Table 2). However, for tasks like visual correspondence, few-shot in-context learning yields competitive results. Interestingly, as the number of shots increases, no consistent performance trend emerges across the different retrieval methods. Moreover, we do not observe significant or consistent performance differences between the retrieval strategies.

Table 12 shows the results of few-shot in-context learning with LLaVA-OneVision-7B on Blink. Here, we find that performance on almost all subtasks is not significantly better than random guessing, even for tasks like object counting and spatial relations, where this model performs much better in baseline settings. Further examination of the outputs reveals that the positional biases identified in Section C persist and even worsen with few-shot prompting, as the model tends to predict the first option in most cases.

In conclusion, while few-shot in-context learning can be effective for some visual perception tasks with GPT-4o, it does not consistently outperform zero-shot baselines and can sometimes negatively impact performance. Additionally, retrieval strategies based on different embedding spaces do not show a clear advantage. For the open-source VLM LLaVA-OneVision-7B, few-shot in-context learning offers no noticeable benefits on these tasks and may even amplify existing biases, further degrading performance.

F Exploring the Importance of Multiple Image Inputs to VLMs

Understanding the relationships between multiple images is crucial for certain visual perception tasks and real-world applications. However, only a few closed-source VLMs (Reid et al., 2024) and a very limited number of open-source VLMs natively support multiple image inputs. For models that do not support this feature, the common practice is to concatenate multiple images into a single image with added margins and input this combined image into the VLM. To investigate this problem, we conduct experiments using concatenated images for tasks requiring multiple image inputs, utilizing both GPT-4o and LLaVA-OneVision-7B. As shown in Table 10, we observe a noticeable decline in performance for both models when multiple images are concatenated into a single image. This decline is particularly consistent with GPT-4o, indicating that concatenating images introduces challenges that these VLMs struggle to handle effectively. This suggests that native support for multiple image inputs is important for maintaining performance, and concatenating images is not the ideal practice for VLMs.

G Detailed VIPACT Algorithm and Function Definitions

This section presents Algorithm 1, which outlines the complete process of our VIPACT framework, providing a clearer illustration of its workflow. To facilitate a comprehensive understanding, Table 15 summarizes the detailed explanations of the functions used in the algorithm. Each function plays a crucial role in orchestrating the interactions between the orchestrator agent, specialized agents, and vision expert models within the VIPACT framework.

Algorithm 1 VipAct: **VI**sual-**P**erception via VLM **A**gent **C**ollaboration & **T**ool-use

```
Require: Set of visual inputs $\mathcal{V}$, a query $q$, a vision-language model $\mathcal{M}$, a set of tools $\mathcal{T} = \{T_1, \ldots, T_n\}$
    including specialized agents and vision expert models, and the maximum iterations $K$
Ensure: An answer $a$ to the visual perception task
Initialize orchestrator agent $\mathcal{O}$ with $\mathcal{M}$ and $\mathcal{T}$
$\mathcal{P}_0 \leftarrow$ FormatPrompt($\mathcal{V}, q$)    ▷ Format initial prompt with visual inputs and query
$t \leftarrow 0$,  $\mathcal{S} \leftarrow \emptyset$    ▷ Initialize iteration counter and state
while $t < K$ and not IsTerminated($\mathcal{S}$) do
    if $\exists T_i \in \mathcal{T}$ : IsRequired($T_i, \mathcal{S}$) then    ▷ Check if any tool is required
        $T^* \leftarrow \arg\max_{T_i \in \mathcal{T}}$ Utility($T_i, \mathcal{S}$)    ▷ Select most useful tool
        $\mathcal{O}_t \leftarrow$ Execute($T^*, \mathcal{S}$)    ▷ Execute selected tool with the current state as input
        if ContainsVisualData($\mathcal{O}_t$) then
            $\mathcal{V} \leftarrow \mathcal{V} \cup$ ProcessVisualData($\mathcal{O}_t$)    ▷ Add new visual data if needed
        end if
    else
        $\mathcal{R}_t \leftarrow \mathcal{M}(\mathcal{P}_{t-1})$    ▷ Generate VLM output
        $\mathcal{O}_t \leftarrow$ InterpretOutput($\mathcal{R}_t$)    ▷ Interpret VLM output
    end if
    $\mathcal{P}_t \leftarrow$ UpdatePrompt($\mathcal{P}_{t-1}, \mathcal{O}_t$)    ▷ Update prompt with new information
    $\mathcal{S} \leftarrow$ UpdateState($\mathcal{S}, \mathcal{O}_t$); $t \leftarrow t + 1$    ▷ Update state with new observations
end while
$a \leftarrow$ ExtractAnswer($\mathcal{S}$)    ▷ Extract final answer from state
return $a$
```

Method (# of shots)	**Sim**	**Count**	**Depth**	**Jig**	**Fun.C**	**Sem.C**	**Spat**	**Local**	**Vis.C**	**Multi-v**
Randomly Choose One From the Options										
Random	50.00	25.00	50.00	50.00	25.00	25.00	50.00	50.00	25.00	50.00
Randomly Select In-context Exemplars										
1-shot	65.93	25.00	71.77	64.00	60.77	56.12	45.45	61.48	86.05	48.12
2-shot	42.22	25.83	73.39	62.00	58.46	58.99	47.55	58.20	88.37	55.64
3-shot	52.59	26.67	51.61	64.00	57.69	57.55	47.55	60.66	88.37	45.11
4-shot	64.44	21.67	66.13	61.33	54.62	55.40	46.85	61.48	88.37	50.38
5-shot	56.30	30.00	70.16	61.33	60.77	59.71	49.65	59.84	87.79	53.38
Select In-context Exemplars Based on CLIP Embedding Similarity										
1-shot	78.52	20.00	66.13	52.00	56.15	56.12	44.06	58.20	87.79	51.88
2-shot	60.00	30.00	61.29	60.67	54.62	53.96	47.55	63.11	84.88	51.88
3-shot	52.59	26.67	59.68	66.00	59.23	54.68	46.15	61.48	89.53	51.13
4-shot	57.04	31.67	68.55	66.00	55.38	56.12	45.45	63.11	88.95	56.40
5-shot	60.00	25.00	64.52	62.67	58.46	53.24	47.55	59.84	87.21	54.89
Select In-context Exemplars Based on VIT Embedding Similarity										
1-shot	73.33	21.67	66.94	55.33	56.15	49.64	46.85	56.56	91.28	48.87
2-shot	63.70	28.33	62.10	60.00	57.69	51.80	47.55	63.93	88.37	52.63
3-shot	57.78	27.50	62.90	64.67	57.69	53.24	46.85	60.66	89.53	50.38
4-shot	46.67	30.83	61.29	64.67	56.92	53.24	48.25	59.02	89.53	48.87
5-shot	54.07	30.00	66.13	68.00	60.77	51.08	45.45	61.40	87.79	51.13

Table 11: Few-shot in-context learning results on the Blink dataset using GPT-4o, evaluated with varying numbers of exemplars and three retrieval methods.

H In-depth Analysis of Visual Input

To better understand the impact of visual input on the performance of our framework, we conducted an ablation study comparing three input conditions:

1. **Image input only.**
2. **Textual description input only**, generated us-

Method (# of shots)	Sim	Count	Depth	Jig	Fun.C	Sem.C	Spat	Local	Vis.C	Multi-v
Randomly Choose One From the Options										
Random	50.00	25.00	50.00	50.00	25.00	25.00	50.00	50.00	25.00	50.00
Randomly Select In-context Exemplars										
1-shot	47.41	13.33	52.42	44.67	21.54	32.37	41.96	43.44	29.65	44.36
2-shot	47.41	2.50	54.03	52.00	22.31	32.37	38.46	43.44	29.65	55.64
3-shot	47.41	5.83	53.23	52.67	22.31	32.37	48.95	43.44	29.65	44.36
4-shot	47.41	3.33	52.42	52.00	22.31	32.37	45.45	43.44	29.65	44.36
5-shot	47.41	17.50	54.84	50.67	22.31	30.94	45.45	43.44	29.65	44.36
Select In-context Exemplars Based on CLIP Embedding Similarity										
1-shot	47.41	8.33	56.45	51.33	21.54	28.06	39.16	43.44	24.42	45.11
2-shot	47.41	8.33	54.84	51.33	22.31	25.18	39.86	43.44	27.91	30.08
3-shot	47.41	10.83	53.23	50.67	20.77	26.62	39.16	43.44	27.33	28.57
4-shot	47.41	10.83	52.42	51.33	23.08	29.50	39.86	43.44	27.91	33.83
5-shot	47.41	11.67	52.42	52.67	20.77	28.06	39.86	43.44	24.42	35.34
Select In-context Exemplars Based on VIT Embedding Similarity										
1-shot	47.41	8.33	56.45	51.33	21.54	28.06	37.06	43.44	24.42	14.29
2-shot	47.41	8.33	54.84	50.67	22.31	25.18	38.46	43.44	27.91	30.08
3-shot	47.41	10.83	53.23	50.67	20.77	26.62	39.86	43.44	27.33	28.57
4-shot	47.41	10.00	52.42	50.67	23.08	29.50	39.86	43.44	27.91	28.57
5-shot	47.41	10.83	52.42	52.00	20.77	28.06	41.96	43.44	24.42	34.59

Table 12: Few-shot in-context learning results on the Blink dataset using LLaVa-OneVision-7B, evaluated with varying numbers of exemplars and three retrieval methods.

Condition	Sim	Count	Depth	Jig	Fun.C	Sem.C	Spat	Local	Vis.C	Multi-v	Overall
Image	81.48	70.00	90.80	68.00	61.50	60.40	86.70	63.11	91.28	62.63	73.59
No image or description input	77.78	59.71	69.35	61.33	53.85	51.08	83.22	60.66	78.49	48.12	64.36
Description	79.32	62.72	73.45	62.37	54.02	52.46	83.22	61.34	81.35	51.42	66.17
Image + description	81.48	70.48	90.80	67.52	62.45	61.32	84.32	62.67	91.28	63.34	73.57

Table 13: Performance comparison across different input conditions using LLaVa-OneVision-7B on the Blink dataset.

Method	Sim	Count	Depth	Jig	Fun.C	Sem.C	Spat	Local	Vis.C	Multi-v
MM-ReAcT (with same tools)	-	30.00	8.69	-	-	-	63.64	5.41	-	-
ViperGPT (with same tools)	-	29.17	3.01	-	-	-	48.95	23.57	-	-
VisProg (with same tools)	-	3.33	5.31	-	-	-	31.47	18.92	-	-
VIPACT	**81.48**	**70.00**	**90.80**	**68.00**	**61.50**	**60.40**	**86.70**	**63.11**	**91.28**	**62.63**

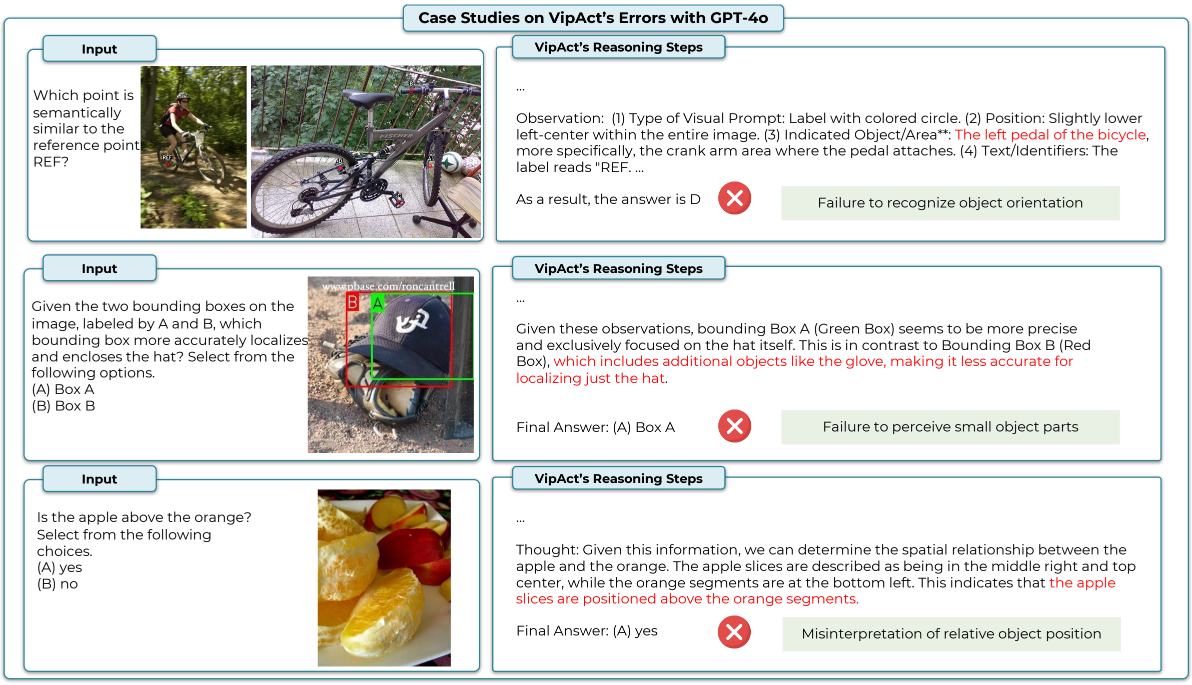
Table 14: Performance comparison of baseline methods integrated with the same vision expert tools as VIPACT. VIPACT significantly outperforms these baselines due to its cohesive multi-agent design and structured integration of vision expert models.

ing GPT-4o with the prompt: “Please generate a detailed description of the following image [IMAGE].”

3. **Both image and textual description input.**

The results, presented in Table 13, indicate that relying solely on textual descriptions leads to lower performance compared to using image input. Although combining both image and textual descriptions provides slight improvements in some cases, the overall performance gain remains minimal. This suggests that textual descriptions alone may fail to capture critical fine-grained visual details and background elements necessary for accurate task execution.

Further analysis reveals that textual descriptions often emphasize objects while overlooking contextual elements, leading to potential biases and suboptimal decision-making in downstream tasks. In contrast, direct image input allows the orchestra-


Case Studies on VipAct's Errors with GPT-4o
Input
VipAct's Reasoning Steps
REF
A
B
C
D
FISCHER
www.pbase.com/roncantrell
B
A
Failure to recognize object orientation
Failure to perceive small object parts
Misinterpretation of relative object position

Which point is semantically similar to the reference point REF?

...

Observation: (1) Type of Visual Prompt: Label with colored circle. (2) Position: Slightly lower left-center within the entire image. (3) Indicated Object/Area**: The left pedal of the bicycle, more specifically, the crank arm area where the pedal attaches. (4) Text/Identifiers: The label reads "REF. ...

As a result, the answer is D ✗

Given the two bounding boxes on the image, labeled by A and B, which bounding box more accurately localizes and encloses the hat? Select from the following options.
(A) Box A
(B) Box B

...

Given these observations, bounding Box A (Green Box) seems to be more precise and exclusively focused on the hat itself. This is in contrast to Bounding Box B (Red Box), which includes additional objects like the glove, making it less accurate for localizing just the hat.

Final Answer: (A) Box A ✗

Is the apple above the orange? Select from the following choices.
(A) yes
(B) no

...

Thought: Given this information, we can determine the spatial relationship between the apple and the orange. The apple slices are described as being in the middle right and top center, while the orange segments are at the bottom left. This indicates that the apple slices are positioned above the orange segments.

Final Answer: (A) yes ✗

Figure 2: Types of error cases in VIPACT with their corresponding reasoning steps.

tor agent to accurately analyze spatial relationships and identify key visual features required for precise task planning.

These findings emphasize the importance of incorporating direct visual input for robust and contextually grounded reasoning. While textual descriptions can supplement visual input, they are insufficient as standalone inputs for tasks requiring fine-grained perception and reasoning.

I Analysis of Tool Set Fairness in Comparisons

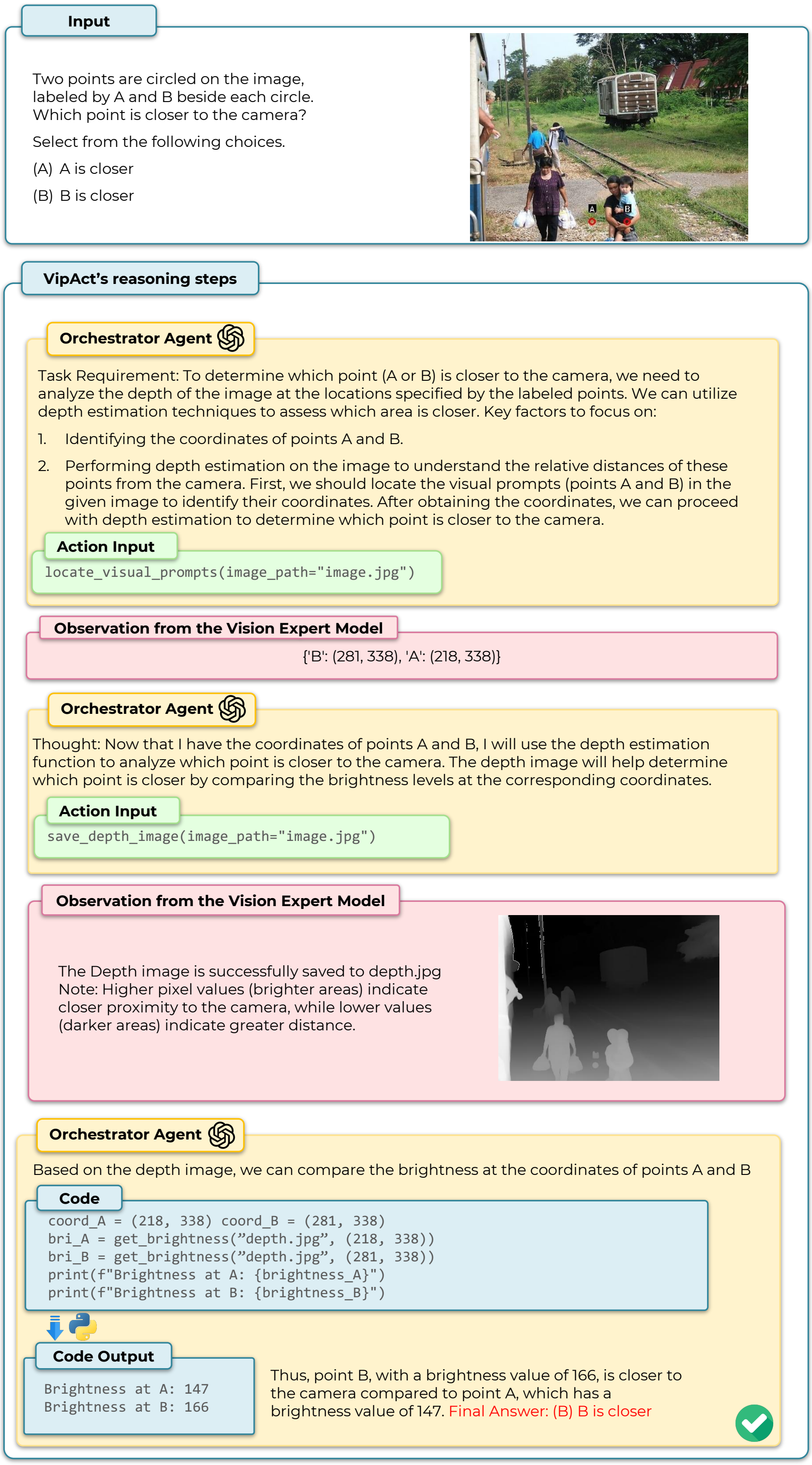
We conducted additional experiments to evaluate the fairness of tool comparisons by integrating our vision expert models into baseline frameworks while preserving their original logic. Unlike many existing visual programming methods, which tightly couple tool usage with internal logic, VIPACT's modular design allows for seamless plug-and-play tool integration with minimal effort, requiring only the definition of a standard Python function header.

Table 14 presents the results of these experiments. While incorporating our vision expert tools into baseline frameworks resulted in minor performance improvements in tasks such as depth estimation and object localization, the overall performance of these methods remained significantly lower than VIPACT. This disparity arises from the persistent limitations in the underlying logic of these frameworks, which hinder their ability to effectively leverage additional tools.

These results highlight the advantages of VIPACT's cohesive design, which combines multi-agent collaboration and a structured approach to integrating vision expert models. By leveraging these elements, VIPACT achieves superior performance and robust generalization across diverse tasks, even when compared to enhanced versions of baseline methods.

J Prompt Design

In this section, we present the complete prompt designs used in our experiments, including the Initial Prompt for the orchestrator agent and the distinct prompt designs for the three specialized agents described in Section 3.



Input

Two points are circled on the image, labeled by A and B beside each circle. Which point is closer to the camera?

Select from the following choices.

(A) A is closer

(B) B is closer

VipAct's reasoning steps

Orchestrator Agent

Task Requirement: To determine which point (A or B) is closer to the camera, we need to analyze the depth of the image at the locations specified by the labeled points. We can utilize depth estimation techniques to assess which area is closer. Key factors to focus on:

1. Identifying the coordinates of points A and B.
2. Performing depth estimation on the image to understand the relative distances of these points from the camera. First, we should locate the visual prompts (points A and B) in the given image to identify their coordinates. After obtaining the coordinates, we can proceed with depth estimation to determine which point is closer to the camera.

Action Input

```
locate_visual_prompts(image_path="image.jpg")
```

Observation from the Vision Expert Model

```
{'B': (281, 338), 'A': (218, 338)}
```

Orchestrator Agent

Thought: Now that I have the coordinates of points A and B, I will use the depth estimation function to analyze which point is closer to the camera. The depth image will help determine which point is closer by comparing the brightness levels at the corresponding coordinates.

Action Input

```
save_depth_image(image_path="image.jpg")
```

Observation from the Vision Expert Model

The Depth image is successfully saved to depth.jpg
Note: Higher pixel values (brighter areas) indicate closer proximity to the camera, while lower values (darker areas) indicate greater distance.

Orchestrator Agent

Based on the depth image, we can compare the brightness at the coordinates of points A and B

Code

```
coord_A = (218, 338) coord_B = (281, 338)
bri_A = get_brightness(”depth.jpg”, (218, 338))
bri_B = get_brightness(”depth.jpg”, (281, 338))
print(f"Brightness at A: {brightness_A}")
print(f"Brightness at B: {brightness_B}")
```

Code Output

```
Brightness at A: 147
Brightness at B: 166
```

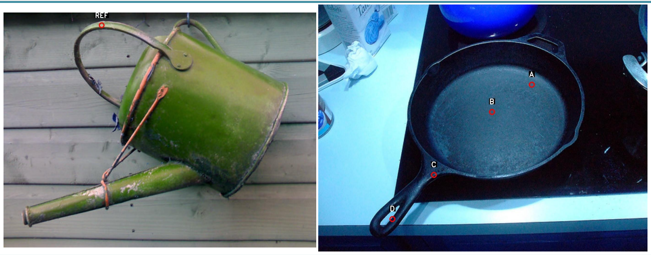
Thus, point B, with a brightness value of 166, is closer to the camera compared to point A, which has a brightness value of 147. Final Answer: (B) B is closer ✓

Figure 3: Case study of the complete reasoning process in VIPACT, incorporating a depth estimation model to assist the VLM in achieving the correct answer.

Input

Given two images with multiple red-circled points labeled A, B, C, and D, select the corresponding point for the reference point (labeled REF) in the first image, considering the action "Pour."

Select from: (A) Point A (B) Point B (C) Point C (D) Point D



VipAct's reasoning steps

Orchestrator Agent

To determine the corresponding point on the second image (a frying pan) for the reference point labeled "REF" on the first image (a watering can) when considering the action "Pour," we need to analyze the functional parts of both objects. Specifically, we should focus on the parts of each object that are used to hold and pour. The reference point on the watering can is located on the handle, which is used to hold the can while pouring. Therefore, we need to identify the point on the frying pan that serves a similar function (i.e., the handle of the frying pan). Key factors to focus on:

1.The location and function of the reference point on the watering can.
2.The corresponding functional part on the frying pan that serves the same purpose for the action "Pour."
3.The visual prompts (red-circled points) on the frying pan and their labels (A, B, C, D).

Thought: To identify the corresponding point on the frying pan, we should first describe the visual prompts in the second image to understand the context of each labeled point (A, B, C, D). This will help us determine which point corresponds to the handle of the frying pan.

Action Input

```
visual_prompt_describe(image_path="images_1.jpg)
```

Observation from the Specialized Agent

The provided image showcases a cast iron skillet with four specific regions or objects indicated by red circles labeled A, B, C, and D. Here's a detailed analysis of each indicated region:
Region A: Location: Inside the skillet, toward the upper right quadrant. 2. Appearance: Smooth, slightly shiny black surface, showing wear due to frequent usage. Context: Part of the main cooking surface, adjacent to the sidewalls. Prompt Type: Red circle labeled "A."
Region B: Location: Near the center of the skille. Appearance: Rougher and matte finish with visible wear, indicating frequent use. Context: Positioned centrally on the main cooking area. Prompt Type: Red circle labeled "B."
Region C: Location: At the base of the skillet handle where it meets the pan. Appearance: Slightly recessed with a noticeable seam between the handle and the pan. Context: A transition area between the pan and handle, showing wear from frequent handling. Prompt Type: Red circle labeled "C."
Region D: Location: Near the end of the skillet's handle. Appearance: Smoother, polished handle with an oblong hole at the end for hanging. Context: Handle area, likely used for holding the skillet. Prompt Type: Red circle labeled "D."

Orchestrator Agent

Thought: Based on the detailed descriptions of the visual prompts in the second image, we now need to determine which point corresponds to the reference point on the first image. The reference point on the watering can is located on the handle, used for holding while pouring. Thus, we need to identify the point on the frying pan that serves a similar function—the handle of the frying pan. From the descriptions, both Point C and Point D are located on the handle of the frying pan. However, Point D, being at the end of the handle, is more analogous to the reference point on the watering can, as it is the part typically held while pouring.
Final Answer: (D) Point D ✓

Figure 4: Case study of the complete reasoning process in VIPACT, incorporating a specialized agent to assist the VLM in achieving the correct answer.

Initial Prompt for Orchestrator Agent

You are a helpful AI agent and please answer the following question based on the image. You have access to the following tools:
{tools}
Additionally, if you want to use python code, you can use the following functions:

```
def image_comparison(image_paths: list, focus: str = None):
        '''
        Compares multiple images and generates a detailed
        analysis of their similarities and differences,
        with an optional focus on specific objects, elements,

        or aspects.

        Parameters
        ----------
        image_paths : list
            A list of file paths for the input images to
            be compared.
        focus : str, optional
            The specific objects, elements, or aspects that
            the comparison should focus on.

            If None, a general comparison is generated.

        Example
        --------
            >>> image_comparison(image_paths=["image1.jpg",
            "image2.jpg"], focus="the cars")
        '''
```

Initial Prompt for Orchestrator Agent (Cont’d)

```
def image_captioning(image_path: str, focus: str = None):
        '''
        Generates a detailed caption for the provided image,


        with an optional focus on specific objects, elements or
        other perspectives that are directly related to solving
        the problem.

        Parameters

        ----------
        image_path : str
            The file path of the input image.
        focus : str, optional
            The specific objects or elements that the caption


            should focus on. If None, a general caption is
            generated.

        Example
        --------


            >>> image_captioning(image_path=”image.jpg”)
            >>> image_captioning(image_path=”image.jpg”,
            focus=”a red car and the background buildings”)
        '''



    def visual_prompt_describe(image_path: str = ”image.jpg”):
        '''
        Analyzes the provided image and describes the specific
        locations and characteristics of various visual prompts



        This function uses a language model to generate a
        detailed description of visual prompts present in the
        image, such as colored circles, bounding boxes, arrows,
        highlights, or textual labels.



        Parameters
        ----------
        image_path : str
            The file path of the input image.



        Example
        --------
            >>> visual_prompt_describe(image_path=”image.jpg”)
```

Initial Prompt for Orchestrator Agent (Cont'd)

```
def save_depth_image(image_path: str = "image.jpg",
    saved_path: str = "depth.jpg"):
        '''

        Estimates the depth of an input image, saves the
        resulting depth image to a specified path,
        and prints out the saved path in a structured format.

        Note: In the processed depth estimation image, brighter


        areas represent objects closer to the camera,
        while darker areas represent objects farther from the
        camera. For pixel values, higher values (brighter areas)
        indicate closer proximity to the camera, while lower
        values (darker areas) indicate greater distance.


        Parameters
        ----------
        image_path : str, optional
            The file path of the input image.


        saved_path : str, optional
            The file path where the resulting depth image will
            be saved. You should make sure the saved image is
            in the same directory as the input image.


        Example
        --------
            >>>  save_depth_image(image_path = "image.jpg",
            saved_path = "depth.jpg")


        '''

    def locate_visual_prompts(image_path: str = "image.jpg"):
        '''
        Analyzes the provided image to identify and accurately


        locate two specific regions labeled 'A' and 'B'.
        This function detects the visual prompts of red circles
        and print out their coordinates.

        Parameters

        ----------
        image_path : str
            The file path of the input image to be processed.

        Example

        -------
            >>> locate_visual_prompts("images/image.jpg")
        '''
```

Initial Prompt for Orchestrator Agent (Cont’d)

```
def compute_clip_similarity(image_path1: str,
   image_path2: str) -> float:
       '''

       Computes the cosine similarity between the CLIP
       embeddings of two images.

       Parameters
       ----------


       image_path1 : str
           The file path of the first input image.
       image_path2 : str
           The file path of the second input image.



       Returns
       -------
       float
           The cosine similarity score between the two image
           embeddings (-1 to 1).


       Example
       -------
           >>> similarity =
           compute_clip_similarity("image1.jpg", "image2.jpg")


       '''

   def segment_image(image_path: str, save_path: str = None)
   -> str:
       '''

       Segments the input image using the SAM model and
       returns the path to the processed image.

       Parameters
       ----------


       image_path : str
           The file path of the input image to be segmented.
       save_path : str, optional
           The file path where the segmented image will be
           saved. If None, a default path is used.


       Returns
       -------
       str
           The file path of the saved segmented image.


       Example
       -------
           >>> segmented_img_path = segment_image("input.jpg",
           "segmented.jpg")
```

Initial Prompt for Orchestrator Agent (Cont'd)

All function implementations are available in the execution environment and you can just call the function without the need to define it.

MUST strictly use the following format:
Question: [The input question you must answer]
Image: [The path of the image, which you may use in external tools]

Task Requirement: [You should provide a comprehensive analysis of the criteria to choose between each option. Include key factors to focus on in solving this task, such as specific visual elements, data points, trends, patterns, and any contextual information that might influence the decision. You can also try to decompose the problem into several key subproblems, with clues inferred from the following steps.]
Thought: [Your reasoning about the question or the last iteration's observations. You should prioritize to think about which tools to use (and which parameters to input) and if you believe no existing tools will help further, use your own knowledge to reason towards the final answer. If there is no observation from the last iteration's tool calling, you should examine the format of tool calling and recall the tool with proper format]
Action Input: [MUST be some of the functions above within a Python block with nothing else. You should figure out which function to use and what are the input parameters.]

Observation: [The output of the called function.]
... (Repeat Thought/Action/Action Input/Observation as needed, you may need to call the tools multiple times if there are multiple images in the input) Thought: [Your final reasoning based on all information gathered]
Final Answer: [You MUST provide a clear answer from the above options without any ambiguity. If a perfect answer is not available, you MUST select the closest possible option.]

Begin! Let's work on the following question! Please remember NOT to estimate any coordinates in the image within the code.
Question: {question}

Image: {image} Task Requirement: (you should start to generate this to begin the iterations)

Prompt for Focused Image Captioning Agent

Please analyze the provided image and generate a comprehensive, detailed caption that focuses specifically on "{focus}".
Your caption should:
1. Identify and describe the specified focus objects or elements in the image, including:

- Quantity (the total number of such object)
- Appearance (color, size, shape, texture)
- Position within the image
- Relation to other objects (if applicable)

2. For the focus objects or elements, describe any actions or events taking place, involving any of them.
3. Mention the overall setting or background of the image, especially in relation to the focus.
4. Include relevant details about lighting, shadows, and any visible textures.

5. If there are people or animals in the focus area, describe their appearances, poses, and any visible expressions.
Your goal is to create an extremely detailed and thorough caption that gives a complete understanding of the image's content with an emphasis on the specified focus, as if you're describing it to someone who cannot see it. Don't leave out any visible elements related to the focus, no matter how minor they might seem.

Image: {image}

Prompt for Focused Image Comparison Agent

Please analyze the provided images and generate a comprehensive, detailed comparison that focuses specifically on "{focus}". Your comparison should:
1. Identify and describe the specified focus (focus) in all images, including:

- Presence or absence in each image (if applicable)
- Quantity (if applicable)
- Position within each image
- Relation to other objects (if applicable)

2. Compare the overall setting or background of the images, but only as it relates to the focus.
2. Summarize the similarities and differences of the focus elements across all images.
3. Describe any changes in actions, events, or states related to the focus elements (if applicable).

5. Analyze differences in lighting, shadows, and visible textures that affect the focus elements.
Your goal is to create a detailed and thorough comparison that gives a complete understanding of how the specified focus elements differ or remain similar across all the provided images. Concentrate primarily on the focus area and only mention other elements if they directly relate to or impact the focus.

Organize your comparison in a clear, structured manner, addressing the focus area in each image in turn and then providing an overall summary of the similarities and differences.
Image: {image}

Prompt for Visual Prompt Description Agent

Please analyze the provided image, emphasizing the specific regions or objects indicated by visual prompts such as colored circles, bounding boxes, arrows, highlights, or textual labels. The most critical aspect of your analysis should be a detailed description of these indicated areas. For each visual prompt:

1. Most importantly, provide an extremely detailed description of the exact region or object being indicated. This is the primary focus of your analysis. Include:
   - Precise location within the larger object or scene
   - Comprehensive details about its appearance (color, texture, shape, size)
   - Any unique features or characteristics
   - Its context and relationship to surrounding elements
2. The type of visual prompt used (e.g., circle, box, arrow, highlight, label).
3. The position of the prompt within the entire image (e.g., top left, center, bottom right).
4. Any text or identifiers associated with the prompt (e.g., labels like ’A’, ’B’, numbers, or short descriptions). Remember, the most crucial part of your response should be the in-depth description of the specific region or object highlighted by each prompt. Provide enough detail that someone could understand exactly what part of the image is being emphasized without seeing the visual prompt itself.

Ensure your description of these indicated regions is as comprehensive as possible, covering every relevant visual aspect. Your goal is to provide a thorough understanding of the highlighted areas, allowing others to easily grasp the significance of each visual prompt in the image.
Image: {image}

Prompt for Few-shot In-context Learning

{The general instruction for the task}
Here are some examples:

Images: {example_images}
Question: {example_question}
Answer: {example_answer}

...
Let’s try another case!
Images: {images}
Question: {question}
Answer:

Function	**Description**
FORMATPROMPT($\mathcal{V}, q$)	Combines the visual inputs $\mathcal{V}$ and the query q into a structured prompt suitable for the vision-language model $\mathcal{M}$. This ensures that the orchestrator agent receives a well-organized task description for reasoning.
ISTERMINATED($\mathcal{S}$)	Checks whether the termination condition has been met based on the current state $\mathcal{S}$. This involves checking for a termination indicator (e.g., `Final Answer:`) or determining if the maximum number of iterations K has been reached.
ISREQUIRED($T_i, \mathcal{S}$)	Determines if a specific tool T_i (either a specialized agent or vision expert model) is necessary given the current state $\mathcal{S}$. This involves checking whether tool-use indicators (e.g., `Action:` or `Action Input:`) have been generated, guiding the orchestrator agent on whether external tools need to be invoked.
UTILITY($T_i, \mathcal{S}$)	Implicitly evaluates the utility of tool T_i in the current context defined by state $\mathcal{S}$. This process involves the orchestrator agent select the most beneficial tool for the next action, based on prior evidence and reasoning steps.
EXECUTE($T^*, \mathcal{S}$)	Executes the selected tool T^* using the current state $\mathcal{S}$ (arguments extracted from VLM's output at this step) as input. The tool processes the input and returns relevant information, such as image data or analytical results, which are then integrated into the reasoning process.
CONTAINSVISUALDATA($\mathcal{O}_t$)	Checks whether the output $\mathcal{O}_t$ from the executed tool includes visual data (e.g., new images or annotations). If visual data is present, it is further processed and incorporated into the reasoning workflow.
PROCESSVISUALDATA($\mathcal{O}_t$)	Processes new visual data from the tool's output $\mathcal{O}_t$ and integrates it into the existing set of visual inputs $\mathcal{V}$. This involves updating the prompt with new image paths to ensure that the visual data is available for subsequent analysis and reasoning.
INTERPRETOUTPUT($\mathcal{R}_t$)	Interprets the output $\mathcal{R}_t$ generated by the VLM $\mathcal{M}$. This step involves converting the raw output into a structured format through rule-based string manipulation, enabling the orchestrator agent to update the task state and inform the next steps.
UPDATEPROMPT($\mathcal{P}_{t-1}, \mathcal{O}_t$)	Updates the current prompt $\mathcal{P}_{t-1}$ with new information derived from the tool output $\mathcal{O}_t$. The updated prompt ensures that the next iteration of the VLM has access to the most recent and relevant context, presented in an organized format for accurate reasoning in the next iteration.
UPDATESTATE($\mathcal{S}, \mathcal{O}_t$)	Updates the current state $\mathcal{S}$ by incorporating new observations and data from the tool or VLM output $\mathcal{O}_t$. This continuous state update allows the system to track progress and adjust its strategy dynamically.
EXTRACTANSWER($\mathcal{S}$)	Extracts the final answer a from the final output of the VLM. This step uses rule-based string matching to retrieve the final prediction from the agent's workflow.

Table 15: Function Definitions in Algorithm 1