OPENVLHARNESS: REAL WORLD VISUAL AGENTIC TOOL-USE WITH MULTIMODAL MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal large language models (MLLMs) have made rapid progress on visual reasoning, yet many tasks require evidence that is difficult to obtain through internal reasoning alone—our pilot study on 12 benchmarks shows that simply increasing the inference budget yields only modest gains—such tasks demand grounded perception, external knowledge, and numerical measurements obtained through heterogeneous tools, whose outputs must be made interpretable to the model and retained across reasoning steps. We introduce OpenVLHarness, a multimodal reasoning harness organized around a capability layer and persistent multimodal memory. The capability layer combines task-level tool interfaces with evidence rendering, providing consistent access to heterogeneous backends and converting their outputs into model-readable observations. Persistent multimodal memory retains the corresponding images, structured data, and web references as addressable artifacts throughout each query. A coding interface executes generated programs over stored data, enabling computations that combine results from multiple tools. Together, these mechanisms support an iterative process of acquiring, inspecting, and composing multimodal evidence. Evaluated on 23 benchmarks across perception, spatial reasoning, visual search, and multimodal question answering, OpenVLHarness consistently surpasses strong MLLM backbones and outperforms existing visual tool-use baselines across six open and proprietary backbones of different scales, including +11.2 with Qwen3-VL-8B, +9.2 with Kimi K3, and +11.2 with GPT-6 Sol. In addition, our controlled study on 12 benchmarks shows that even though OpenVLHarness incurs additional cost from multi-step tool use, giving the base GPT-6 Luna an even larger reasoning budget (1.27$\times$ the cost of OpenVLHarness) improves the average only from 49.6 to 51.5, compared with 62.9 for OpenVLHarness. These results suggest that additional reasoning alone does not substitute for acquiring and composing task-relevant multimodal evidence.

1 INTRODUCTION

Multimodal large language models (MLLMs) are increasingly used as reasoning engines over images, documents, webpages, and videos. However, many real-world visual tasks require information that the model cannot reliably infer from the input alone. Answering a question may require localizing numerous tiny objects in dense scenes, retrieving external knowledge about a long-tail entity, or computing quantities from intermediate measurements. Such tasks require additional multimodal evidence acquired through perception, search, and computation. Our inference-time scaling study (Fig. 1c) reveals diminishing returns at higher reasoning budgets: moving the base model from xhigh to max more than doubles model-token spending but improves its average score by only 3.0 points. This raises the question of whether further progress requires better access to task-relevant multimodal evidence, beyond additional reasoning over the initial input.

Visual tool-use agents address this need by connecting MLLMs to perception models, search engines, and code execution (Hong et al., 2026a; Wu et al., 2025a; Narayan et al., 2025; Geng et al., 2026; Su et al., 2025; Wu et al., 2025b; Surís et al., 2023; Gupta & Kembhavi, 2023). Building a general harness around these tools presents two connected challenges. First, heterogeneous backends expose different calling conventions and return outputs such as segmentation masks, depth maps, and retrieved webpages that need to be presented in a form the model can interpret. Second, subsequent reasoning steps may still need access to the underlying artifacts: a rendered depth

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

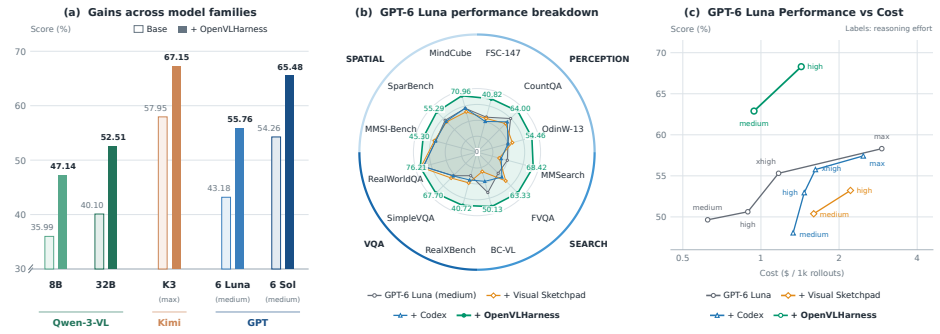


Figure 1: **Performance of OpenVLHarness across models and harnesses.** **(a)** Average score over our 23 benchmarks with and without OpenVLHarness, across the Qwen, Kimi, and GPT model families. **(b)** Per-benchmark comparison with the base model, Visual Sketchpad, and Codex, using GPT-6 Luna at medium reasoning effort. **(c)** Average score versus cost on 12 benchmarks with GPT-6 Luna across reasoning-effort settings.

map supports visual inspection, but a precise measurement requires its numerical values. Effective multimodal tool use therefore requires both consistent tool interfaces with evidence rendering and a mechanism for retaining and reusing the resulting artifacts.

We introduce OpenVLHarness, a multimodal reasoning harness organized around a *capability layer* and *persistent multimodal memory*. The capability layer combines *task-level tool interfaces* with *model-facing evidence rendering*. Task-level interfaces define operations independently of a particular backend, allowing tools to be replaced without changing the orchestrator's task vocabulary. Evidence rendering converts heterogeneous outputs into observations that combine visualizations, concise descriptions, and compact structured information. For example, visual grounding accepts an image and a textual query, then presents the detected regions through an annotated image and coordinates. Together, these mechanisms let the orchestrator invoke tools consistently and inspect their results in a form suited to multimodal reasoning.

Persistent multimodal memory makes the corresponding artifacts available across turns within a query. Images, structured data, and web references are registered under reusable handles, allowing later calls to operate on earlier evidence. In the grounding example, the orchestrator can inspect the overlay, then reuse the retained masks and a subsequently acquired depth array to measure depth within each region. A Python Coding Agent supports such compositions: the orchestrator specifies a computational objective in natural language, and a code-generation session backed by the same backbone model writes a program that executes over the stored data. The capability layer makes tool results interpretable; memory makes their underlying artifacts addressable and reusable. Together, they support an iterative process of acquiring, inspecting, and computing over multimodal evidence.

We evaluate OpenVLHarness on 23 benchmarks spanning perception, spatial reasoning, visual search, and general visual question answering, using six open and proprietary MLLM backbones of different scales (Fig. 1a). Across these settings, OpenVLHarness consistently improves average performance over the base MLLMs by 9.2-12.6 percentage points and outperforms existing visual tool-use baselines (Fig. 1b). Scaling experiments on 12 benchmarks further show that OpenVLHarness also benefits from greater reasoning effort (Fig. 1c), suggesting that evidence acquisition and inference-time scaling are complementary. We conduct ablation studies to examine the incremental effects of the capability layer and memory, showing the value of both usable tool outputs and retained multimodal artifacts. Lastly, for users who would like to adapt our harness to a custom domain, we further showcase two coding-agent-driven workflows for specialization: harness search edits prompts, tool descriptions, and output templates, while tool generation composes existing capabilities into new tools. Both operate without updating the underlying MLLM weights.

Our contributions are: **(1)** a capability layer that combines task-level tool interfaces with model-facing evidence rendering for heterogeneous perception, search, and computation tools; **(2)** persistent multimodal memory that retains addressable artifacts across turns and enables their reuse in subsequent tool calls and memory-grounded computation; and **(3)** an evaluation across 23 benchmarks and six backbones, with component ablations and domain-specialization studies, accompanied by an open-source release of the harness, tools, evaluation protocols, and two optimization workflows.

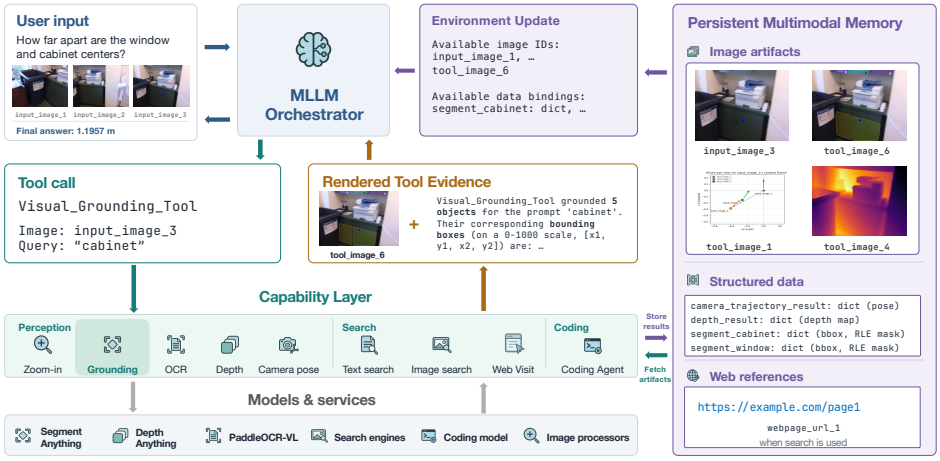


Figure 2: **Overview of OpenVLHarness.** The MLLM orchestrator invokes capabilities through typed interfaces. The capability layer connects these interfaces to underlying models and services and renders their outputs into model-facing evidence: here, a labeled mask overlay accompanied by a textual description and normalized bounding boxes. Persistent multimodal memory retains image artifacts, structured data, and web references under reusable handles. An Environment Update advertises the available artifacts to the orchestrator, allowing subsequent calls to reuse earlier evidence, including through memory-grounded computation.

2 Harness Design

OpenVLHarness augments an MLLM orchestrator with two complementary mechanisms (Figure 2). The *capability layer* combines task-level tool interfaces with model-facing evidence rendering, providing consistent access to tools and observations the orchestrator can inspect (Sec. 2.1). *Persistent multimodal memory* retains the corresponding artifacts and exposes them by reference for reuse across turns within a query (Sec. 2.2). At each step, the orchestrator uses the observations and available memory references to select a tool call or produce an answer. A Python Coding Agent supports computation over stored artifacts, connecting outputs from different capabilities into multi-step procedures. The modular interfaces and renderers also support harness specialization through text edits and tool composition (Sec. 3.5).

2.1 Capability Layer

We design the capability layer with two goals: defining how the orchestrator invokes a capability and rendering the resulting evidence for inspection. The first accommodates different backends behind a consistent interface; the second addresses the heterogeneous formats of their outputs.

Task-level tool interfaces. We define a capability by the operation it provides, its typed inputs, and the evidence and reusable data it exposes. For example, visual grounding accepts an image reference and a textual query and exposes object locations, associated metadata, and a visualization; a supporting backend may additionally provide segmentation masks. Each backend is wrapped to implement this task-level interface. Visual Grounding can thus use SAM 3 or Grounding DINO, and OCR can use PaddleOCR-VL or DeepSeek-OCR, while preserving the orchestrator's calling convention (Appendix C). Backend selection can reflect accuracy, cost, and latency requirements. Our base instantiation includes nine capabilities: five for perception—Zoom-in, Visual Grounding, OCR, Depth Estimation, and Camera Trajectory; three for information seeking—Text Search, Image Search, and Webpage Visit; and a Python Coding Agent for computation over intermediate results. Please refer to Appendix C and Table 15 for their implementation details.

Model-facing evidence rendering. Raw tool outputs often require additional processing to become useful observations. An RLE mask encodes a region precisely, but serializing that encoding into the context does not show the orchestrator which object was segmented. Dense numerical arrays can likewise consume substantial context without providing useful, interpretable evidence.

Each capability therefore includes an output processor that renders its result into a model-facing observation. Depending on the capability, this observation combines visual attachments, concise natural-language descriptions, compact structured information, and references to reusable artifacts. Grounding can return a labeled mask overlay with the number of detected objects and normalized bounding boxes; depth estimation can return a colorized map; and camera tracking can return a visualization of the recovered trajectory. The underlying masks, arrays, and camera parameters are passed to memory for later reuse. Figure 2 illustrates grounding for "cabinet" in `input_image_3`. The capability returns a mask overlay with a description of the five grounded objects and their bounding boxes, allowing the orchestrator to inspect the detections. If a subsequent step requires a depth measurement for each cabinet, this rendered observation supplies the visual context, while the retained masks and a depth array supply the data for computation. The memory mechanism below connects these steps.

2.2 Persistent Multimodal Memory

The capability layer makes tool results interpretable; persistent multimodal memory makes their underlying artifacts addressable and reusable. A later tool call must be able to identify a particular image, mask, or array and access its contents. Repeating a description of a previous result does not provide this access.

We maintain an explicit artifact registry for each query, with entries retained across its reasoning trajectory. The registry separates an artifact's payload from the handle used to reference it. The orchestrator sees available names and compact descriptors, while the harness retains the underlying values and resolves references when needed. The registry contains three categories:

- **Image artifacts:** input, tool-generated, and retrieved images, exposed through handles such as `input_image_3` and `tool_generated_image_6`. Duplicate images reuse an existing registry entry.
- **Structured data bindings:** named values containing boxes, scores, masks, depth arrays, camera parameters, OCR results, and other tool outputs. Each binding has a descriptor identifying its fields and types, including array shapes where applicable.
- **Web references:** retrieved URLs exposed through handles such as `webpage_url_1`, allowing later navigation without requiring the model to reproduce the original URL.

Registering and reusing artifacts. After a successful tool call, the harness registers supported artifacts and associates their handles with the capability's rendered observation. Images are delivered as visual attachments with named references; structured payloads remain accessible through data bindings. An *Environment Update* advertises the available image handles, data bindings, and web references after each tool turn, exposing the current namespace without repeating every artifact's full payload. In the cabinet example, the grounding call adds an overlay image and bindings for the detection data, including bounding boxes and RLE masks.

When a subsequent call refers to an artifact, the harness resolves its handle to the input required by the receiving tool. An image handle supplies the image for a later visual operation, a web reference supplies a URL for Webpage Visit, and a data binding supplies values for code execution. The orchestrator can thus reuse the grounding and depth results without reconstructing their contents from earlier descriptions. References connect retained evidence across tools and turns. The detailed representation and handles can be found in Appendix C.4 and Table 16.

Memory-grounded computation. To compute depth within the detected cabinets, the orchestrator delegates a natural-language objective to the Python Coding Agent, such as "compute the average depth within each detected region." The harness supplies descriptors of the available memory bindings to a fresh code-generation context of the same backbone model. The generated program executes with the actual binding values available as named variables, allowing it to combine the grounding masks with the depth array. Execution results and feedback return in language, with supported output artifacts registered for subsequent reuse. The orchestrator can then use the result or revise the objective in a later call.

This mechanism also supports filtering detections, aggregating OCR values, and other computations over intermediate results without requiring the orchestrator to serialize large arrays into its request.

The capability layer supplies reusable operations and interpretable observations; memory and code execution connect their outputs into longer reasoning procedures.

3 EXPERIMENTS

3.1 TASKS, MODELS, AND BASELINES

Tasks. We evaluate OpenVLHarness across four task domains spanning 23 datasets, as summarized in Table 1 (splits, sizes, and scoring in Appendix D):

- **Counting and grounding.** Six benchmarks targeting object counting and open-vocabulary visual grounding: PixmoCount (Deitke et al., 2025), CountQA (Tamarapalli et al., 2025), FSC-147 (Ranjan et al., 2021), OdinW-13 (Li et al., 2022), FSCD-147 (Nguyen et al., 2022), and FSCD-LVIS (Nguyen et al., 2022).
- **Multimodal search and deep research.** Six benchmarks drawn from a suite of 11 multimodal information-seeking tasks: FVQA (Wu et al., 2025a), MMSearch (Jiang et al., 2024), MMSearch-Plus (Tao et al., 2026), HLE-VL (Phan et al., 2025), BrowseComp-VL (Geng et al., 2026), and RealX-Bench (Hong et al., 2026a); the remaining five are evaluated in Appendix B.5.
- **General VQA and hallucination.** Six benchmarks covering general visual question answering and hallucination robustness: RealWorldQA (xAI, 2024), SimpleVQA (Cheng et al., 2025), HallusionBench (Guan et al., 2024), FREAK-VQA (Yin et al., 2026), FREAK-MCQ (Yin et al., 2026), and V* (Wu & Xie, 2024).
- **Spatial understanding.** Five benchmarks targeting spatial and embodied reasoning: MMSI-Bench (Yang et al., 2025b), ERQA (Team et al., 2025), EmbSpatial-Bench (Du et al., 2024), SparBench (Zhang et al., 2026a), and MindCube (Wang et al., 2025c).

Models and baselines. We instantiate OpenVLHarness on four open-source models across different sizes—Qwen2.5-VL-7B-Instruct (Qwen Team, 2025a), Qwen3-VL-8B-Instruct (Qwen Team, 2025b), Qwen3-VL-32B-Instruct, and Kimi K3 (Team et al., 2026)—and three proprietary models: GPT-5 (OpenAI, 2025b), GPT-6 Luna, and GPT-6 Sol (OpenAI, 2026). Per model sampling parameters can be found in Appendix C.6. Because the search APIs, the agent's tool-routing decisions, and the LLM judge are all stochastic, every Qwen3-VL result, for OpenVLHarness and for all baselines, is the mean of *three* independent runs with different random seeds, while results with proprietary backbones are single runs because of API cost (standard deviations in Appendix H). In addition to the no-tool baseline (denoted **Base**), we compare OpenVLHarness against two groups of baselines: **Training-based baselines**: AdaReasoner (Song et al., 2026), PixelReasoner (Wang et al., 2025a), and DeepEyesV2 (Hong et al., 2026a)—harnesses that require fine-tuning the backbone to adapt it to the tool-use framework—evaluated with their released Qwen2.5-VL-7B checkpoints (Appendix Table 8; setup in Appendix G.2). **Training-free baselines**: Visual Sketchpad (Hu et al., 2024) and Octotools (Lu et al., 2025) are visual tool-use harnesses evaluated on the same Qwen3-VL backbones as OpenVLHarness (detailed setup in Appendix G.1). We additionally evaluate Visual Sketchpad and Codex (code mode, with web search disabled) on GPT-6 Luna.

3.2 MAIN RESULTS

Across the evaluated backbones, OpenVLHarness improves average performance over the underlying base model and outperforms all existing training-free harnesses (Table 1). On Qwen3-VL-8B, it raises the 23-dataset average from 36.0 to 47.1 ($+11.2$), exceeding the strongest training-free baseline, Octotools (33.3), by 13.8 points; on Qwen3-VL-32B, the average rises from 40.1 to 52.5 ($+12.4$). The gains extend to GPT-5 ($45.7 \rightarrow 57.2$) and Kimi K3 ($58.0 \rightarrow 67.2$), which achieves the highest overall average among the evaluated backbones (Appendix Table 6). On Qwen2.5-VL-7B backbone, OpenVLHarness averages 35.58 across the 23 datasets, surpassing the training-based baselines AdaReasoner (27.54), PixelReasoner (29.34), and DeepEyesV2 (26.66), without updating model weights (Appendix Table 8).

The GPT-6 results highlight that effective multimodal tool use depends on the harness. With GPT-6 Luna, Codex averages 41.5, below the no-tool baseline (43.2), while OpenVLHarness reaches 55.8. This comparison shows that a coding-agent harness does not automatically improve multimodal

		Qwen-3-VL-8B				Qwen-3-VL-32B				GPT-6 Luna				GPT-6 Sol	
Domain	**Dataset**	**Base**	**OT**	**VS**	**Ours**	**Base**	**OT**	**VS**	**Ours**	**Base**	**VS**	**CX**	**Ours**	**Base**	**Ours**
Counting and Grounding	FSC-147	15.32	11.26	9.41	**37.71**	18.97	12.44	12.35	**39.42**	28.69	27.84	25.74	**40.82**	38.72	**44.40**
	CountQA	32.00	29.06	19.83	**48.56**	44.70	41.75	32.85	**64.14**	54.40	47.51	49.87	**64.00**	73.00	**76.00**
	PixmoCount	74.53	66.60	68.69	**85.21**	79.59	78.75	75.33	**87.64**	78.65	82.96	78.46	**89.70**	89.14	**91.20**
	OdinW-13	39.80	24.60	31.77	**50.40**	42.39	44.33	36.10	**59.13**	33.76	37.76	33.78	**54.46**	40.86	**57.88**
	FSCD-147	1.79	0.93	1.41	**23.52**	2.88	2.55	2.30	**28.45**	5.56	7.07	4.33	**25.61**	8.36	**29.42**
	FSCD-LVIS	1.47	1.36	2.49	**27.37**	1.79	3.91	4.57	**31.63**	2.95	5.84	2.90	**28.96**	5.17	**35.22**
Visual Search and Deep Research	MMSearch	14.04	9.94	14.62	**51.46**	20.47	18.13	18.71	**62.57**	40.94	31.58	33.92	**68.42**	55.56	**77.78**
	MMSearch-Plus	1.29	3.54	4.18	**14.79**	2.57	3.54	5.14	**17.68**	5.79	6.75	11.25	**22.83**	16.72	**45.66**
	RealX-Bench	22.16	26.80	22.16	**34.71**	24.74	26.80	29.90	**40.21**	20.62	25.77	19.59	**40.72**	36.08	**51.55**
	FVQA	28.39	32.67	28.67	**55.00**	32.67	35.00	33.33	**62.67**	36.67	47.33	35.67	**63.33**	58.67	**71.00**
	HLE-VL	6.36	7.58	4.24	**10.30**	10.00	9.09	6.67	**11.52**	25.15	17.27	23.33	**25.45**	36.67	**43.33**
	BC-VL	21.05	15.79	10.53	**32.33**	28.82	19.55	15.04	**33.08**	39.10	21.30	40.85	**50.13**	43.86	**58.90**
General VQA and Hallucination	RealWorldQA	71.37	69.67	69.15	**72.67**	78.82	76.21	74.38	**80.52**	72.81	75.56	72.42	**76.21**	84.58	**87.45**
	SimpleVQA	54.33	46.27	42.81	**70.00**	59.00	51.75	49.58	**70.33**	42.67	46.07	42.96	**67.70**	60.30	**76.40**
	HallusionBench	57.76	55.66	54.01	**59.07**	61.66	53.82	59.38	**63.40**	62.97	**65.85**	59.17	64.23	72.77	**75.27**
	FREAK-VQA	**44.68**	38.80	33.04	37.05	44.06	45.06	37.92	**48.19**	44.81	44.56	42.05	**50.56**	54.94	**64.08**
	FREAK-MCQ	36.42	31.39	35.31	**36.82**	44.06	38.73	38.33	**45.77**	49.40	50.40	45.47	**54.73**	61.77	**70.32**
	V*	**83.24**	73.30	71.73	**83.24**	85.34	76.44	85.86	**88.48**	82.72	76.96	73.82	**90.05**	91.10	**93.19**
Spatial Understanding	MMSI-Bench	31.60	27.90	28.10	**33.00**	35.00	36.60	32.40	**38.00**	36.30	37.20	36.40	**45.30**	49.80	**57.40**
	ERQA	**44.00**	41.25	35.50	43.00	44.75	46.25	**48.25**	48.00	55.25	57.50	51.50	**58.50**	66.50	**71.25**
	EmbSpatial	78.37	73.71	75.74	**78.90**	**80.54**	76.15	78.98	80.00	70.82	73.52	68.19	**74.59**	80.60	**82.53**
	SparBench	39.55	38.32	34.63	**43.73**	**46.98**	45.29	44.69	46.76	45.51	43.66	44.32	**55.29**	55.36	**62.66**
	MindCube	28.36	39.71	41.44	**55.48**	32.40	45.96	44.13	**60.19**	57.50	53.65	57.69	**70.96**	67.40	**83.17**
	Average	35.99	33.31	32.15	**47.14**	40.10	38.61	37.66	**52.51**	43.18	42.78	41.46	**55.76**	54.26	**65.48**

Table 1: **Main results across four task domains and 23 datasets.** Results are grouped by MLLM backbone. **Base** denotes the backbone without tool use; for harness, **OT**, **VS**, and **CX** denote OctoTools, Visual Sketchpad, and Codex, respectively; **Ours** denotes OpenVLHarness. GPT-5 and Kimi K3 results are in Table 6. Bold indicates the best result within each backbone group, and underlining indicates the best result across all shown backbones. Qwen3-VL results are means of three runs (standard deviations in Table 23); GPT-6 results are single runs.

reasoning. With GPT-6 Sol, OpenVLHarness improves every one of the 23 benchmarks, raising the average from 54.3 to 65.5 and demonstrating consistent benefits across all four domains.

Counting, grounding, and multimodal search benefit substantially from dedicated perception and external retrieval. Gains on general VQA and spatial understanding depend more on the backbone: Luna and Sol improve on every benchmark in these domains, with average gains of $+8.0$/$+6.9$ on general VQA and hallucination and $+7.9$/$+7.5$ on spatial understanding, respectively. This pattern is consistent with greater proficiency in using tools and reasoning over their visual evidence. The less consistent gains on Qwen3-VL-8B motivate the failure analysis and domain specialization in Secs. 3.4 and 3.5.

Cost Efficiency. Figure 1c compares GPT-6 Luna across reasoning-effort settings on 12 benchmarks. Base and the evaluated Codex configuration improve with additional effort, but their gains diminish at the higher spending levels. At maximum effort, they reach average scores of 58.3 and 57.4, respectively, below OpenVLHarness at medium effort (62.9), which uses less than half their model-token spending. Increasing the reasoning effort of OpenVLHarness from medium to high further raises its score to 68.3. These results suggest that acquiring and composing multimodal evidence complements stronger reasoning: additional computation remains useful, but does not eliminate the benefit of the harness within the tested budgets. Appendix B.6 provides a complementary cost-controlled comparison and per-dataset results. Reported costs cover model tokens, including reasoning tokens, and exclude tool execution and judging.

Qualitative examples. Figure 3 shows how these gains arise. In each trajectory the agent chains several capabilities and reuses their outputs through memory: camera poses, depth maps, and grounding masks become bindings for exact 3D computation (a); grounded crops are passed by

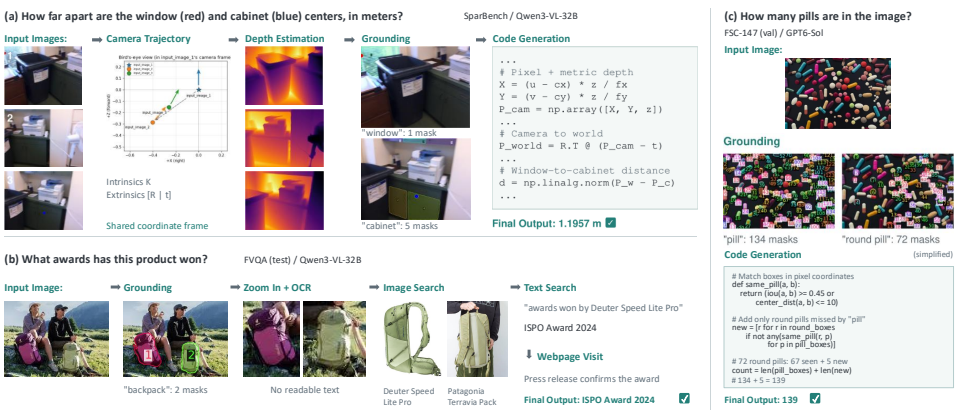


Figure 3: **Qualitative examples of OpenVLHarness. (a)** Camera poses, depth, and grounding are combined in code to measure a 3D distance (Example 17). **(b)** Grounded crops are passed to OCR and image search, and the answer is verified on the web (Example 8). **(c)** Masks from two grounding queries are reconciled in code to reach the exact count (Example 1).

Variant	Counting and Grounding		Visual Search and Deep Research		General VQA and Hallucination		Spatial Understanding		Overall	
	8B	32B	8B	32B	8B	32B	8B	32B	8B	32B
Baseline (no tool)	27.48	31.72	18.40	23.34	57.97	62.16	44.38	47.93	37.06	41.29
+ Tool	41.70	43.30	21.44	24.89	52.41	61.41	47.83	52.84	40.84	45.61
+ Capability layer	46.73	46.20	25.97	32.74	56.05	63.63	48.16	52.63	44.23	48.80
+ Textual memory	**47.27**	45.74	26.92	34.43	57.54	64.84	47.98	51.46	44.93	49.12
OpenVLHarness	45.46	**51.74**	**33.10**	**37.96**	**59.81**	**66.12**	**50.82**	**54.59**	**47.14**	**52.51**

Table 2: **Ablation study of OpenVLHarness across model scales.** We progressively add raw tool access, the capability layer, textual memory, and multimodal memory to the no-tool baseline. Results are reported across four benchmark domains and overall for 8B and 32B models, averaged over three runs; standard deviations are provided in Table 24.

identifier to zoom-in, OCR, and image search before the answer is verified on the web (b); and masks from two grounding queries are reconciled in code to correct an undercount (c).

3.3 Ablation Study

Table 2 examines the two mechanisms introduced in Sec. 2: the *capability layer*, which combines task-level tool interfaces with model-facing evidence rendering, followed by *persistent multimodal memory*, which makes the resulting artifacts addressable and reusable across turns. Starting from the no-tool baseline, we add raw tool access, then the capability layer, and finally memory, using a textual-memory variant as an intermediate comparison (details in Appendix B.2).

Raw tool access raises the overall average by $+3.8$ points at 8B and $+4.3$ at 32B, with the largest gains in counting and grounding. Adding the capability layer restores full tool schemas and renders heterogeneous outputs into concise observations, yielding another $+3.4$ and $+3.2$ points. These gains support the joint value of task-level tool interfaces and model-facing evidence rendering; the comparison does not separate their individual contributions.

Adding textual memory enables structured data bindings, web references, and Environment Updates, but represents tool-generated images only through descriptions, without reusable image handles. This adds $+0.7$ points at 8B and $+0.3$ at 32B. Completing persistent multimodal memory with visual attachments and reusable image handles adds a further $+3.1$ and $+4.5$, bringing the full harness to 47.1 and 52.5. The largest gains from this final step occur in multimodal search and deep research ($+6.2$/$+3.5$), with additional gains in spatial understanding ($+2.8$/$+3.1$). The results support the method's complementary mechanisms: interpretable tool observations, followed by retained multimodal artifacts that can be revisited and composed across turns.

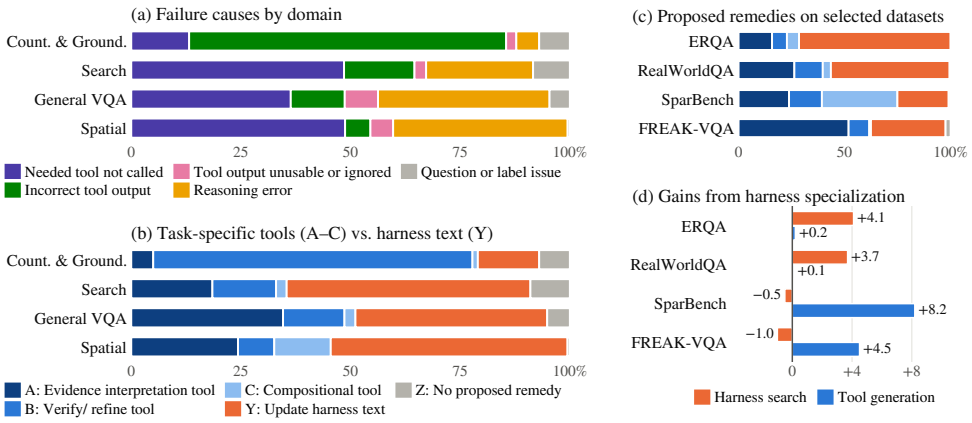


Figure 4: **Failure diagnosis and harness specialization with Qwen3-VL-8B.** We critique failure trajectories using Kimi K3. **(a)** Distribution of failure causes across four domains. **(b)** Proposed remedies: grouped into creating task-specific tools (A–C), updating harness-text (Y), and no proposed remedy (Z). **(c)** Four datasets selected for contrasting remedy profiles: Y (orange) dominates ERQA and RealWorldQA, while A-C (blue) dominates SparBench and FREAK-VQA. **(d)** Gains from harness search and tool generation over the unoptimized harness. Harness search is more effective on datasets dominated by text-level changes, while tool generation yields larger gains when task-specific tools are preferred.

3.4 Failure Analysis

We use Kimi K3 to critique 9,980 failed Qwen3-VL-8B trajectories across 23 benchmarks. Given the question, answer, images, trajectory, and toolbox, the critic assesses each tool call and identifies the trajectory's primary failure cause and proposes a fix. Its labels agree with human annotations on 92% of 100 sampled failures. The protocol, coverage, and taxonomy are detailed in Appendix E.1.

Surprisingly, we find that failure patterns differ across domains (Fig. 4a). Despite substantial overall gains, counting and grounding remain challenging in crowded images with more than 25 objects, where missed or merged instances limit exhaustive detection. Such perception errors account for the dominant "tool output wrong" category (72% of failures in this domain). Search failures most often involve an omitted tool call (49%), while failures in spatial understanding and general VQA involve both omitted tool calls and reasoning errors, consistent with our hypothesis that the model has limited proficiency in spatial reasoning and tool use. Failures involving unusable or ignored correct tool outputs are uncommon (2–8% across domains), suggesting that the capability layer generally presents retrieved evidence in a usable form.

Failure causes alone do not determine how to specialize the harness: an omitted call may require clearer instructions or a new composition of tools. We therefore combine the cause and proposed fix into remedy channels (Fig. 4b). Constructing new task-specific tools can make evidence interpretation explicit (A), enforce systematic verification, filtering of tool results (B), or supply a missing tool composition (C). In contrast, harness text changes (Y)—edits to the system prompt, tool descriptions, and output messages—guide the model to call existing tools when needed and use their results appropriately. The remaining channel (Z) covers cases without a proposed remedy. These diagnostic recommendations guide optimization rather than predict recoverable accuracy, since the critique covers only failed trajectories. Exact mapping rules and per-dataset shares are in Appendix E.2.

3.5 Harness Optimization

When adapting OpenVLHarness to a custom domain, users can use failures on their own data to identify which parts of the harness need refinement. We provide two coding-agent-driven workflows for specializing the general-purpose harness without updating model weights. **Harness search** edits system prompts, tool descriptions, and evidence-rendering templates, selecting candidates through repeated evaluation. **Tool generation** implements and refines task-specific tools by composing existing capabilities (algorithms in Appendix E.3). To compare them, we select four datasets with contrasting remedy profiles (Fig. 4c): harness text changes (Y) dominate ERQA (71%) and Real-

WorldQA (56%), whereas task-specific tools (A+B+C) dominate SparBench (75%) and FREAK-VQA (62%). We run both workflows on all four datasets with Qwen3-VL-8B, testing whether their relative benefits align with these profiles.

The results follow this distinction (Fig. 4d). Harness search improves ERQA and RealWorldQA by $+4.1$ and $+3.7$ points, compared with $+0.2$ and $+0.1$ from tool generation. Conversely, tool generation improves SparBench and FREAK-VQA by $+8.2$ and $+4.5$, while harness search yields -0.5 and -1.0. The generated tools compute metric 3D relations on SparBench and make visibility decisions explicit on FREAK, matching the missing-composition and evidence-interpretation needs identified by the critique (Appendix Table 21). These results suggest a practical approach to adaptation on custom data: diagnose the prevailing failures, then use harness search to improve the use of existing capabilities or tool generation to add the task-specific operations the domain requires.

4 Related Works

Visual Tool-Use Baselines. Early visual tool-use works most notably treat the problem as program synthesis, compiling a question into a sequential plan that invokes vision tools in a pipeline to produce an answer (Surís et al., 2023; Gupta & Kembhavi, 2023; Kamali & Kordjamshidi, 2026; Zhao et al., 2025; Wu et al., 2023; Yang et al., 2023b; Gao et al., 2023; Castrejon et al., 2024; Zhang et al., 2026b; Zhou et al., 2024; Menon et al., 2024; Chen et al., 2024b; You et al., 2023); because these plans are fixed in place, they cannot be adjusted based on tool execution results. A later line adopts multi-turn frameworks that interleave reasoning with visual feedback, but depends on brittle, task-specific framework designs that require extensive training to work (Ma et al., 2025; Zhou et al., 2025a; Song et al., 2026; Wang et al., 2025a; Zheng et al., 2025; Hong et al., 2026a; Su et al., 2025; Wu et al., 2025b; Zhao et al., 2026), and many domain-specific visual tool-use frameworks fall into the same category (Chen et al., 2025a; Wu et al., 2025a; Narayan et al., 2025; Geng et al., 2026; Huang et al., 2026; Yao et al., 2026; Yang et al., 2026; Du et al., 2026; Wang et al., 2025d). Closest to our setting, Visual Sketchpad (Hu et al., 2024) and Octotools (Lu et al., 2025) are training-free tool-use frameworks built around manually designed tool interfaces and execution strategies. In contrast, we focus on making the harness itself more adaptable, so its evidence processing and tool composition can be specialized as tools, MLLMs, and target tasks evolve.

Working Memory for Tool-Using Agents Language agents maintain intermediate information to support reasoning and action across multiple steps. Context management approaches further separate compact model-visible state from more detailed external information, enabling the agent to selectively recover relevant content when needed (Packer et al., 2023; Lee et al., 2024). Tool-using agents have also introduced explicit mechanisms for passing intermediate results across actions. ReWOO (Xu et al., 2023) represents tool evidence through referenceable variables, while CodeAct (Wang et al., 2024a) maintains intermediate results as variables in an executable interpreter. HuggingGPT (Shen et al., 2023) and LLMCompiler (Kim et al., 2024) go further in using symbolic references to intermediate tool outputs that are resolved when executing dependent downstream calls. Building on these ideas, OpenVLHarness extends to the multimodal setting, maintaining a unified registry of typed artifacts spanning images, structured data, and web evidence. It exposes compact model-facing handles while preserving the underlying artifacts for later inspection, composition, and reuse through modality-appropriate downstream tools.

5 Conclusion

We presented OpenVLHarness, a general harness for real-world visual agentic tool use built around a capability layer and persistent multimodal memory. The capability layer combines task-level tool interfaces with evidence rendering, enabling the orchestrator to invoke heterogeneous tools and inspect their outputs. Persistent multimodal memory retains the corresponding artifacts for reuse across turns, including computation over stored data through a Python Coding Agent. Together, these mechanisms support reasoning that acquires and composes multimodal evidence. Evaluation across 23 benchmarks and six backbones demonstrates broad gains, while component ablations show the incremental benefits of the capability layer and multimodal memory. Harness search and tool generation further specialize the system on selected domains without updating the underlying MLLM weights. We will release the harness, tools, evaluation protocols, and optimization workflows to support further work on multimodal tool-use agents.

AI USE STATEMENT

In this work, we used generative AI tools to design or provide feedback on research methodology and experiments, implement methods, assist with translation, support qualitative and thematic data analysis, and interpret results. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for or assist in the writing of mathematical proofs, propose or refine hypotheses, or clean and reformat datasets; the remaining required disclosure tasks are not applicable to this work.

Additionally, we used generative AI tools to create or modify scientific figures or images, suggest experimental parameters, create or edit software code, draft parts of the research paper, summarize or analyze existing literature, discover research topics or identify gaps, source and search for information, edit the manuscript to improve readability, identify relevant literature, suggest the structure of the paper, and propose titles or keywords. We did not use generative AI tools to formulate questions for surveys or interviews, transcribe recordings of research material, or format references.

All AI-assisted work was reviewed by the authors. AI-assisted methodological and experimental suggestions were evaluated by the authors before being incorporated into the study, and AI-generated or edited code was inspected and tested for correctness through our experimental pipeline. Suggestions related to literature were checked against the original sources, and interpretations of experimental results were verified against the underlying experimental outputs. AI-assisted figures and manuscript text were reviewed and revised by the authors for accuracy, clarity, and consistency with the reported methods and results. We take responsibility for the final content of this work, including text, claims, code, figures, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects or the collection of sensitive personal data. Our experiments are conducted on existing research benchmarks and are intended to study general-purpose multimodal reasoning and tool use. We are not aware of additional ethical concerns specific to this work beyond those generally associated with the use and deployment of large multimodal models.

REPRODUCIBILITY STATEMENT

We provide the source code in the supplementary materials, with detailed setup and reproduction instructions in the accompanying `README.md`. Section 2 describes the harness design, with tool implementations, memory representations, and the system prompt detailed in Appendix C. Benchmark splits and evaluation protocols are documented in Appendix D, model configurations and sampling parameters in Appendix C.6, and baseline reproduction settings in Appendix G. Appendix E provides the failure critique protocol, specialization algorithms, and examples of optimized harness text and generated tools.

REFERENCES

Manoj Acharya, Kushal Kafle, and Christopher Kanan. TallyQA: Answering complex counting questions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Lakshya A Agrawal et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Niki Amini-Naieni, Tengda Han, and Andrew Zisserman. CountGD: Multi-modal open-world counting. In *Advances in Neural Information Processing Systems*, 2024.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Zhongang Cai, Yubo Wang, Qingping Sun, Ruisi Wang, Chenyang Gu, Wanqi Yin, Zhiqian Lin, Zhitao Yang, Chen Wei, Xuanke Shi, Kewang Deng, Xiaoyang Han, Zukai Chen, Jiaqi Li, Xiangyu Fan, Hanming Deng, Lewei Lu, Bo Li, Ziwei Liu, Quan Wang, Dahua Lin, and Lei Yang. Holistic evaluation of multimodal llms on spatial intelligence. *arXiv preprint arXiv:2508.13142*, 2025.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, Jie Lei, Tengyu Ma, Baishan Guo, et al. SAM 3: Segment anything with concepts. *arXiv preprint arXiv:2511.16719*, 2025.

Lluis Castrejon, Thomas Mensink, Howard Zhou, Vittorio Ferrari, Andre Araujo, and Jasper Uijlings. Hammr: Hierarchical multimodal react agents for generic vqa. *arXiv preprint arXiv:2404.05465*, 2024.

Jierun Chen, Fangyun Wei, Jinjing Zhao, Sizhe Song, Bohuai Wu, Zhuoxuan Peng, S.-H. Gary Chan, and Hongyang Zhang. Revisiting referring expression comprehension evaluation in the era of large multimodal models. *arXiv preprint arXiv:2406.16866*, 2024a.

Shuang Chen, Kaituo Feng, Hangting Chen, Wenxuan Huang, Dasen Dai, Quanxin Shou, Yunlong Lin, Xiangyu Yue, Shenghua Gao, and Tianyu Pang. OpenSearch-VL: An open recipe for frontier multimodal search agents. *arXiv preprint arXiv:2605.05185*, 2026.

Siyi Chen, Mikaela Angelina Uy, Chan Hee Song, Faisal Ladhak, Adithyavairavan Murali, Qing Qu, Stan Birchfield, Valts Blukis, and Jonathan Tremblay. Spacetools: Tool-augmented spatial reasoning via double interactive rl. *arXiv preprint arXiv:2512.04069*, 2025a.

Yang Chen, Hexiang Hu, Yi Luan, Haitian Sun, Soravit Changpinyo, Alan Ritter, and Ming-Wei Chang. Can pre-trained vision and language models answer visual information-seeking questions? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Zhenfang Chen, Rui Sun, Wenjun Liu, Yining Hong, and Chuang Gan. Genome: generative neuro-symbolic visual reasoning by growing and reusing modules. In *International Conference on Learning Representations*, volume 2024, pp. 2012–2039, 2024b.

Zhenfang Chen, Delin Chen, Rui Sun, Wenjun Liu, and Chuang Gan. Scaling autonomous agents via automatic reward modeling and planning. In *International Conference on Learning Representations*, volume 2025, pp. 29237–29268, 2025b.

Xianfu Cheng, Wei Zhang, Shiwei Zhang, Jian Yang, Xiangyuan Guan, Xianjie Wu, Xiang Li, Ge Zhang, Jiaheng Liu, Yuying Mai, Yutao Zeng, Zhoufutu Wen, Ke Jin, Baorui Wang, Weixiao Zhou, Yunhong Lu, Tongliang Li, Wenhao Huang, and Zhoujun Li. SimpleVQA: Multimodal factuality evaluation for multimodal large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4637–4646, 2025. arXiv:2502.13059.

Gheorghe Comanici et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 91–104, 2025.

Nikola Djukic, Alan Lukezic, Vitjan Zavrtanik, and Matej Kristan. A low-shot object counting network with iterative prototype adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

Mengfei Du, Binhao Wu, Zejun Li, Xuan-Jing Huang, and Zhongyu Wei. Embspatial-bench: Benchmarking spatial understanding for embodied tasks with large vision-language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 346–355, 2024.

Yifan Du, Zikang Liu, Jinbiao Peng, Jie Wu, Junyi Li, Jinyang Li, Wayne Xin Zhao, and Ji-Rong Wen. Towards long-horizon agentic multimodal search. *arXiv preprint arXiv:2604.12890*, 2026.

Haodong Duan, Junming Yang, Yuxuan Qiao, Xinyu Fang, Lin Chen, Yuan Liu, Xiaoyi Dong, Yuhang Zang, Pan Zhang, Jiaqi Wang, et al. Vlmevalkit: An open-source toolkit for evaluating large multi-modality models. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 11198–11201, 2024.

Difei Gao, Lei Ji, Luowei Zhou, Kevin Qinghong Lin, Joya Chen, Zihan Fan, and Mike Zheng Shou. Assistgpt: A general multi-modal assistant that can plan, execute, inspect, and learn. *arXiv preprint arXiv:2306.08640*, 2023.

Xinyu Geng, Peng Xia, Zhen Zhang, Xinyu Wang, Qiuchen Wang, Ruixue Ding, Chenxi Wang, Jialong Wu, Yida Zhao, Kuan Li, Huifeng Yin, Yong Jiang, Pengjun Xie, Fei Huang, Huaxiu Yao, Yi R. Fung, and Jingren Zhou. WebWatcher: Breaking new frontiers of vision-language deep research agent. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. arXiv:2508.05748.

Tianrui Guan, Fuxiao Liu, Xiyang Wu, Ruiqi Xian, Zongxia Li, Xiaoyu Liu, Xijun Wang, Lichang Chen, Furong Huang, Yaser Yacoob, et al. Hallusionbench: an advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14375–14385, 2024.

Agrim Gupta, Piotr Dollar, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5356–5364, 2019.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14953–14962, 2023.

Kimihiro Hasegawa, Wiradee Imrattanatrai, Masaki Asada, Ken Fukuda, and Teruko Mitamura. Tama: Tool-augmented multimodal agent for procedural activity understanding. *arXiv preprint arXiv:2510.00161*, 2025.

Jack Hong, Chenxiao Zhao, ChengLin Zhu, Weiheng Lu, Guohai Xu, and Xing Yu. DeepEyesV2: Toward agentic multimodal model. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026a.

Yining Hong, Rui Sun, Bingxuan Li, Xingcheng Yao, Maxine Wu, Alexander Chien, Da Yin, Ying Nian Wu, Zhecan James Wang, and Kai-Wei Chang. Embodied web agents: Bridging physical-digital realms for integrated agent intelligence. *arXiv preprint arXiv:2506.15677*, 2025.

Yining Hong, Huang Huang, Manling Li, Li Fei-Fei, Jiajun Wu, and Yejin Choi. Learning from trials and errors: Reflective test-time planning for embodied llms. *arXiv preprint arXiv:2602.21198*, 2026b.

Wenbo Hu, Jingli Lin, Yilin Long, Yunlong Ran, Lihan Jiang, Yifan Wang, Chenming Zhu, Runsen Xu, Tai Wang, and Jiangmiao Pang. G^2VLM: Geometry grounded vision language model with unified 3d reconstruction and spatial reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9535–9546, 2026.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *Advances in Neural Information Processing Systems*, 37:139348–139379, 2024.

Wenxuan Huang, Yu Zeng, Qiuchen Wang, Zhen Fang, Shaosheng Cao, Zheng Chu, Qingyu Yin, Shuang Chen, Zhenfei Yin, Lin Chen, Zehui Chen, Xu Tang, Yao Hu, Shaohui Lin, Philip Torr, Feng Zhao, and Wanli Ouyang. Vision-DeepResearch: Incentivizing DeepResearch capability in multimodal large language models. *arXiv preprint arXiv:2601.22060*, 2026.

Dongzhi Jiang, Renrui Zhang, Ziyu Guo, Yanmin Wu, Jiayi Lei, Pengshuo Qiu, Pan Lu, Zehui Chen, Chaoyou Fu, Guanglu Song, et al. Mmsearch: Benchmarking the potential of large models as multi-modal search engines. *arXiv preprint arXiv:2409.12959*, 2024.

Eric Hanchen Jiang, Zhi Zhang, Yuchen Wu, Levina Li, Dong Liu, Xiao Liang, Rui Sun, Yubei Li, Edward Sun, Haozheng Luo, et al. Memory as a controlled process: Learned adaptive memory management for llm agents. *arXiv preprint arXiv:2607.13591*, 2026.

Danial Kamali and Parisa Kordjamshidi. Neptune: A neuro-pythonic framework for tunable compositional reasoning on vision-language. In *International Conference on Learning Representations*, volume 2026, pp. 86124–86148, 2026.

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

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. In *The Twelfth International Conference on Learning Representations*, 2024.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24370–24391. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/kim24y.html.

Kimi Team. Kimi K3 technical report. https://platform.kimi.ai/docs/guide/kimi-k3-quickstart, 2026.

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

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John F. Canny, and Ian Fischer. A human-inspired reading agent with gist memory of very long contexts. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 26396–26415. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/lee24c.html.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10965–10975, 2022.

Rang Li, Lei Li, Shuhuai Ren, Hao Tian, Shuhao Gu, Shicheng Li, Zihao Yue, Yudong Wang, Wenhan Ma, Zhe Yang, et al. Groundingme: Exposing the visual grounding gap in mllms through multi-dimensional evaluation. *arXiv preprint arXiv:2512.17495*, 2025a.

Shilong Li, Xingyuan Bu, Wenjie Wang, Jiaheng Liu, Jun Dong, Haoyang He, Hao Lu, Haozhe Zhang, Chenchen Jing, Zhen Li, Chuanhao Li, Jiayi Tian, Chenchen Zhang, Tianhao Peng, Yancheng He, Jihao Gu, Yuanxing Zhang, Jian Yang, Ge Zhang, Wenhao Huang, Wangchunshu Zhou, Zhaoxiang Zhang, Ruizhe Ding, and Shilei Wen. MM-BrowseComp: A comprehensive benchmark for multimodal browsing agents. *arXiv preprint arXiv:2508.13186*, 2025b.

Haotong Lin, Sili Chen, Junhao Liew, Donny Y. Chen, Zhenyu Li, Guang Shi, Jiashi Feng, and Bingyi Kang. Depth anything 3: Recovering the visual space from any views. *arXiv preprint arXiv:2511.10647*, 2025.

Chang Liu, Yujie Zhong, Andrew Zisserman, and Weidi Xie. CounTR: Transformer-based generalised visual counting. In *British Machine Vision Conference*, 2022.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection. In *European Conference on Computer Vision*, 2024.

Pan Lu, Bowen Chen, Sheng Liu, Rahul Thapa, Joseph Boen, and James Zou. Octotools: An agentic framework with extensible tools for complex reasoning. *arXiv preprint arXiv:2502.11271*, 2025.

Zixian Ma, Jianguo Zhang, Zhiwei Liu, Jieyu Zhang, Juntao Tan, Manli Shu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Caiming Xiong, et al. Latte: Learning to think with vision specialists. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 11203–11240, 2025.

Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan L. Yuille, and Kevin Murphy. Generation and comprehension of unambiguous object descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Sachit Menon, Richard Zemel, and Carl Vondrick. Whiteboard-of-thought: Thinking step-by-step across modalities. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 20016–20031, 2024.

Thomas Mensink, Jasper Uijlings, Lluis Castrejon, Arushi Goel, Felipe Cadar, Howard Zhou, Fei Sha, Andre Araujo, and Vittorio Ferrari. Encyclopedic VQA: Visual questions about detailed properties of fine-grained categories. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Kartik Narayan, Yang Xu, Tian Cao, Kavya Nerella, Vishal M. Patel, Navid Shiee, Peter Grasch, Chao Jia, Yinfei Yang, and Zhe Gan. DeepMMSearch-R1: Empowering multimodal LLMs in multimodal web search. *arXiv preprint arXiv:2510.12801*, 2025.

Thanh Nguyen, Chau Pham, Khoi Nguyen, and Minh Hoai. Few-shot object counting and detection. In *European Conference on Computer Vision*, pp. 348–365. Springer, 2022.

OpenAI. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

OpenAI. Codex CLI: A lightweight coding agent that runs in your terminal. `https://github.com/openai/codex`, 2025a. Version 0.155.1.

OpenAI. OpenAI GPT-5 system card. *arXiv preprint arXiv:2601.03267*, 2025b.

OpenAI. Introducing GPT-6 Sol and GPT-6 Luna. `https://openai.com/index/introducing-gpt-6-sol-and-luna/`, 2026.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. *ArXiv*, 2023.

PaddlePaddle Team. PaddleOCR-VL: Boosting multilingual document parsing via a 0.9B ultra-compact vision-language model. *arXiv preprint arXiv:2510.14528*, 2025.

Roni Paiss, Ariel Ephrat, Omer Tov, Shiran Zada, Inbar Mosseri, Michal Irani, and Tali Dekel. Teaching CLIP to count to ten. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Qwen Team. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025a.

Qwen Team. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025b.

Viresh Ranjan, Udbhav Sharma, Thu Nguyen, and Minh Hoai. Learning to count everything. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3394–3403, 2021.

Baochang Ren, Yunzhi Yao, Rui Sun, Shuofei Qiao, Ningyu Zhang, and Huajun Chen. Aligning agentic world models via knowledgeable experience learning. *arXiv preprint arXiv:2601.13247*, 2026.

Asankhaya Sharma. OpenEvolve: an open-source evolutionary coding agent. `https://github.com/algorithmicsuperintelligence/openevolve`, 2025.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving AI tasks with chatgpt and its friends in hugging face. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL `http://papers.nips.cc/paper_files/paper/2023/hash/77c33e6a367922d003ff102ffb92b658-Abstract-Conference.html`.

Taiwei Shi, Sihao Chen, Bowen Jiang, Linxin Song, Longqi Yang, and Jieyu Zhao. Experiential reinforcement learning. *arXiv preprint arXiv:2602.13949*, 2026.

Mingyang Song, Haoyu Sun, Jiawei Gu, Linjie Li, Luxin Xu, Ranjay Krishna, and Yu Cheng. Adareasoner: Dynamic tool orchestration for iterative visual reasoning. *arXiv preprint arXiv:2601.18631*, 2026.

Zhaochen Su, Linjie Li, Mingyang Song, Yunzhuo Hao, Zhengyuan Yang, Jun Zhang, Guanjie Chen, Jiawei Gu, Juntao Li, Xiaoye Qu, et al. Openthinkimg: Learning to think with images via visual tool reinforcement learning. *arXiv preprint arXiv:2505.08617*, 2025.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11888–11898, 2023.

Jayant Sravan Tamarapalli, Rynaa Grover, Nilay Pande, and Sahiti Yerramilli. Countqa: How well do mllms count in the wild? *arXiv preprint arXiv:2508.06585*, 2025.

Xijia Tao, Yihua Teng, Xinxing Su, Xinyu Fu, Jihao Wu, Chaofan Tao, Ziru Liu, Haoli Bai, Rui Liu, and Lingpeng Kong. MMSearch-Plus: Benchmarking provenance-aware search for multimodal browsing agents. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. arXiv:2508.21475.

Gemini Robotics Team, S Abeyruwan, J Ainslie, JB Alayrac, MG Arenas, T Armstrong, A Balakrishna, R Baruch, M Bauza, M Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Haozhe Wang, Alex Su, Weiming Ren, Fangzhen Lin, and Wenhu Chen. Pixel reasoner: Incentivizing pixel-space reasoning with curiosity-driven reinforcement learning. *arXiv preprint arXiv:2505.15966*, 2025a.

Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. VGGT: Visual geometry grounded transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025b.

Qineng Wang, Baiqiao Yin, Pingyue Zhang, Jianshu Zhang, Kangrui Wang, Zihan Wang, Jieyu Zhang, Keshigeyan Chandrasegaran, Han Liu, Ranjay Krishna, Saining Xie, Jiajun Wu, Li Fei-Fei, and Manling Li. Mindcube: Spatial mental modeling from limited views, 2025c. URL https://arxiv.org/abs/2506.21458.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232. PMLR / OpenReview.net, 2024a. URL https://proceedings.mlr.press/v235/wang24h.html.

Yikun Wang, Zuyan Liu, Ziyi Wang, Han Hu, Pengfei Liu, and Yongming Rao. Geovista: Web-augmented agentic visual reasoning for geolocalization. *arXiv preprint arXiv:2511.15705*, 2025d.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024b.

Haoran Wei, Yaofeng Sun, and Yukun Li. DeepSeek-OCR: Contexts optical compression. *arXiv preprint arXiv:2510.18234*, 2025a.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. BrowseComp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025b.

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

Jinming Wu, Zihao Deng, Wei Li, Yiding Liu, Bo You, Bo Li, Zejun Ma, and Ziwei Liu. MMSearch-R1: Incentivizing LMMs to search. *arXiv preprint arXiv:2506.20670*, 2025a.

Mingyuan Wu, Jingcheng Yang, Jize Jiang, Meitang Li, Kaizhuo Yan, Hanchao Yu, Minjia Zhang, Chengxiang Zhai, and Klara Nahrstedt. Vtool-r1: Vlms learn to think with images via reinforcement learning on multimodal tool use. *arXiv preprint arXiv:2505.19255*, 2025b.

Penghao Wu and Saining Xie. V*: Guided visual search as a core mechanism in multimodal llms. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13084–13094. IEEE, 2024.

xAI. Realworldqa. https://x.ai/news/grok-1.5v, 2024. Introduced in the Grok-1.5 Vision Preview announcement.

Yunzhong Xiao, Yangmin Li, Hewei Wang, Yunlong Tang, and Zora Zhiruo Wang. Toolmem: Enhancing multimodal agents with learnable tool capability memory. *arXiv preprint arXiv:2510.06664*, 2025.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *CoRR*, abs/2305.18323, 2023. doi: 10.48550/ARXIV.2305.18323. URL https://doi.org/10.48550/arXiv.2305.18323.

BY Yan, Chaofan Li, Hongjin Qian, Shuqi Lu, and Zheng Liu. General agentic memory via deep research. *arXiv preprint arXiv:2511.18423*, 2025a.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025b.

Cheng-Fu Yang, Da Yin, Wenbo Hu, Heng Ji, Nanyun Peng, Bolei Zhou, and Kai-Wei Chang. Verbalized representation learning for interpretable few-shot generalization. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1602–1612. IEEE, 2025a.

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

Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Sihan Yang, Runsen Xu, Yiman Xie, Sizhe Yang, Mo Li, Jingli Lin, Chenming Zhu, Xiaochen Chen, Haodong Duan, Xiangyu Yue, et al. Mmsi-bench: A benchmark for multi-image spatial intelligence. *arXiv preprint arXiv:2505.23764*, 2025b.

Zhaorui Yang, Bo Pan, Han Wang, Yiyao Wang, Xingyu Liu, Luoxuan Weng, Yingchaojie Feng, Haozhe Feng, Minfeng Zhu, Bo Zhang, et al. Multimodal deepresearcher: Generating text-chart interleaved reports from scratch with agentic framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 34368–34377, 2026.

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

Huanjin Yao, Qixiang Yin, Min Yang, Ziwang Zhao, Yibo Wang, Haotian Luo, Jingyi Zhang, and Jiaxing Huang. MM-DeepResearch: A simple and effective multimodal agentic search baseline. *arXiv preprint arXiv:2603.01050*, 2026.

Zhihan Yin, Jianxin Liang, Yueqian Wang, Yifeng Yao, Huishuai Zhang, and Dongyan Zhao. Freak: A fine-grained hallucination evaluation benchmark for advanced mllms. *arXiv preprint arXiv:2603.19765*, 2026.

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

Licheng Yu, Patrick Poirson, Shan Yang, Alexander C. Berg, and Tamara L. Berg. Modeling context in referring expressions. In *European Conference on Computer Vision*, 2016.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, et al. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, et al. Learning to discover at test time. *arXiv preprint arXiv:2601.16175*, 2026.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*, 2025.

Jiahui Zhang, Yurui Chen, Yueming Xu, Ze Huang, Jilin Mei, Chunhui Chen, Yanpeng Zhou, Yu-Jie Yuan, Xinyue Cai, Guowei Huang, et al. From flatland to space: Teaching vision-language models to perceive and reason in 3d. *Advances in Neural Information Processing Systems*, 38, 2026a.

Zhehao Zhang, Ryan A Rossi, Tong Yu, Franck Dernoncourt, Ruiyi Zhang, Jiuxiang Gu, Sungchul Kim, Xiang Chen, Zichao Wang, and Nedim Lipka. Vipact: Visual-perception enhancement via specialized vlm agent collaboration and tool-use. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 36536–36546, 2026b.

Zhengbo Zhang, Jinbo Su, Zhaowen Zhou, Changtao Miao, Yuhan Hong, Qimeng Wu, Yumeng Liu, Feier Wu, Yihe Tian, Yuhao Liang, Zitong Shan, Wanke Xia, Yi-Fan Zhang, Bo Zhang, Zhe Li, Shiming Xiang, and Ying Yan. VisBrowse-Bench: Benchmarking visual-native search for multimodal browsing agents. *arXiv preprint arXiv:2603.16289*, 2026c.

Shitian Zhao, Haoquan Zhang, Shaoheng Lin, Ming Li, Qilong Wu, Kaipeng Zhang, and Chen Wei. Pyvision: Agentic vision with dynamic tooling. *arXiv preprint arXiv:2507.07998*, 2025.

Shitian Zhao, Shaoheng Lin, Ming Li, Haoquan Zhang, Wenshuo Peng, Kaipeng Zhang, and Chen Wei. Pyvision-rl: Forging open agentic vision models via rl. *arXiv preprint arXiv:2602.20739*, 2026.

Ziwei Zheng, Michael Yang, Jack Hong, Chenxiao Zhao, Guohai Xu, Le Yang, Chao Shen, and Xing Yu. Deepeyes: Incentivizing" thinking with images" via reinforcement learning. *arXiv preprint arXiv:2505.14362*, 2025.

Qiji Zhou, Ruochen Zhou, Zike Hu, Panzhong Lu, Siyang Gao, and Yue Zhang. Image-of-thought prompting for visual reasoning refinement in multimodal large language models. *arXiv preprint arXiv:2405.13872*, 2024.

Zetong Zhou, Dongping Chen, Zixian Ma, Zhihan Hu, Mingyang Fu, Sinan Wang, Yao Wan, Zhou Zhao, and Ranjay Krishna. Reinforced visual perception with tools. *arXiv preprint arXiv:2509.01656*, 2025a.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*, 2025b.

APPENDIX

A TOOL-USE BEHAVIOR

Table 1 reports the accuracy of OpenVLHarness. This section characterizes the behavior that produces it: how frequently the agent invokes a tool, which capabilities it invokes, and how those invocations relate to accuracy relative to the same model without tools. The analysis covers four base models — Qwen3-VL-8B, Qwen3-VL-32B, GPT-6 Luna and GPT-6 Sol — and reads every rollout of the OpenVLHarness runs behind Table 1, approximately 99,000 rollouts containing 183,000 tool calls. All counts are exact rather than sampled. A question is *engaged* if the agent emitted at least one tool call on it, regardless of whether the call succeeded.

Measurement protocol. Section A.3 compares OpenVLHarness against the **Base** column of Table 1 on a per-question basis: both arms answer the same questions and are scored with the same dataset metric, and differences are averaged over paired rows. This design supports statements about individual capabilities, since the rows on which a given tool was invoked can be isolated and both arms evaluated on that subset. The resulting quantities are correlational rather than causal: the agent selects the rows on which it invokes a tool, and the remaining capabilities are available on those rows as well. Each per-tool measurement is therefore reported alongside a control, namely the same paired difference computed on the rows that did not invoke the tool. A capability whose two values coincide is not, on its own, responsible for the observed gain. The control is a complement set, so its composition depends on how often the capability is invoked: for a capability used on a small fraction of questions the control approximates the model's overall gain, whereas for one used on the majority of questions the control is a small and atypical remainder.

A.1 TOOL-CALL VOLUME AND ENGAGEMENT

Two quantities characterize how heavily a model uses the tools, and they vary independently: the number of calls a question costs, and the fraction of questions answered with no call. Table 3 reports them per model, and Figure 5 resolves them by domain.

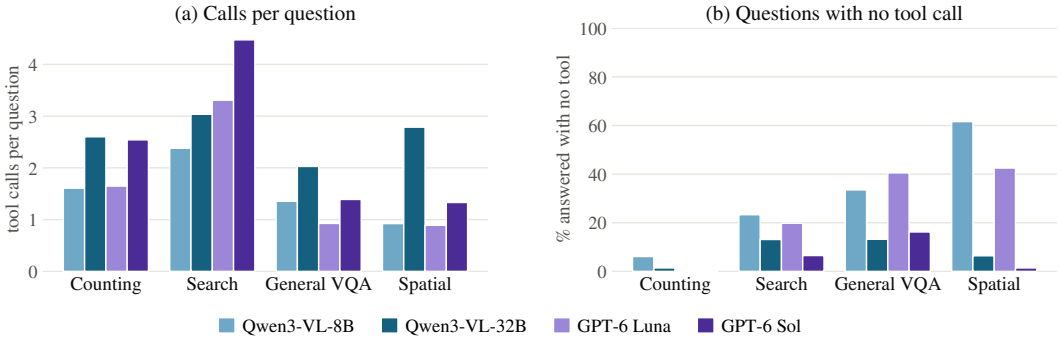


Figure 5: **Tool-call volume and engagement by domain. (a)** Tool calls per question, averaged over all questions in the domain. **(b)** Fraction of questions answered with no tool call. Engagement varies with model scale within a family and not across families, and call volume does not order the models by accuracy.

Volume and engagement. Engagement varies with model scale within a family but not across families. The smaller models leave a substantial fraction of questions untouched (Qwen3-VL-8B 29.0%, GPT-6 Luna 23.5%), whereas the larger models rarely do (6.3% and 4.7%). Call volume does not order the models by accuracy: GPT-6 Luna issues the fewest calls per question of the four and attains higher accuracy than either Qwen model in Table 1. By domain, counting and grounding engages all four models on nearly every question, while spatial reasoning and general VQA account for most of the unengaged questions in the smaller models.

Reliability and context cost. Fewer than 1% of calls return a tool error for any model, so execution reliability is not a limiting factor. Repeated calls, defined as byte-identical invocations within

Model	Calls / question	Calls / engaged question	No tool (%)	Messages / question	Repeat calls (%)	Failed calls (%)
Qwen3-VL-8B	1.41	1.99	29.0	9.7	1.94	0.59
Qwen3-VL-32B	2.55	2.72	6.3	14.2	0.53	0.58
GPT-6 Luna	1.37	1.79	23.5	8.2	0.41	0.75
GPT-6 Sol	2.05	2.15	4.7	10.2	1.92	0.58

Table 3: **Tool-use profile of OpenVLHarness across four base models.** Calls per question averages over all questions, including those answered without tools; calls per engaged question conditions on questions with at least one call. Messages count transcript turns, including the environment update emitted after each call. Repeat calls are byte-identical to an earlier call in the same rollout; failed calls are those the tool server returned as errors.

one rollout, are more frequent than errors but remain a small fraction of traffic and do not scale with model size. The dominant cost is context: a rollout occupies between 8 and 14 transcript turns, of which two to three and a half are environment updates, one emitted at the start and one after each call.

A.2 Distribution of Tool Calls

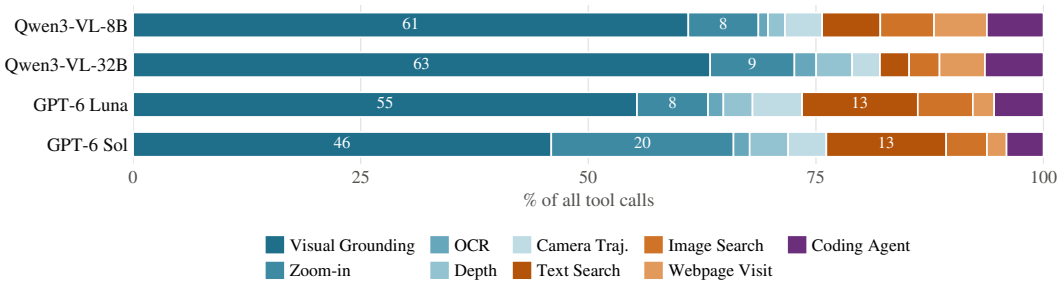


Figure 6: **Distribution of tool calls across capabilities.** All calls issued by each model, normalized to 100%. Visual Grounding is the largest consumer for every model; the models differ mainly in the second-ranked capability.

Aggregate distribution. Visual Grounding accounts for the largest share of calls in all four models, ranging from 46% for GPT-6 Sol to 63% for Qwen3-VL-32B. The models differ mainly in the second-ranked capability. GPT-6 Sol allocates 20% of its calls to Zoom-in, against 8–9% for the other models, corresponding to a ground-then-inspect pattern. The GPT-6 models allocate a larger share to web retrieval (13% to Text Search each), concentrated in the search domain. Depth Estimation, Camera Trajectory and the Python Coding Agent are small in aggregate and concentrated in the spatial datasets.

Distribution by domain. Figure 7 resolves the same budget by domain. Counting and grounding is dominated by Visual Grounding for every model, at 90–92% of domain calls, with the exception of GPT-6 Sol, which allocates roughly a quarter of its calls in that domain to Zoom-in. Search is the only domain in which the three web capabilities dominate. Spatial reasoning is the one domain in which four capabilities carry comparable load, combining grounding, camera trajectory, depth and the coding agent, and general VQA divides between grounding and inspection.

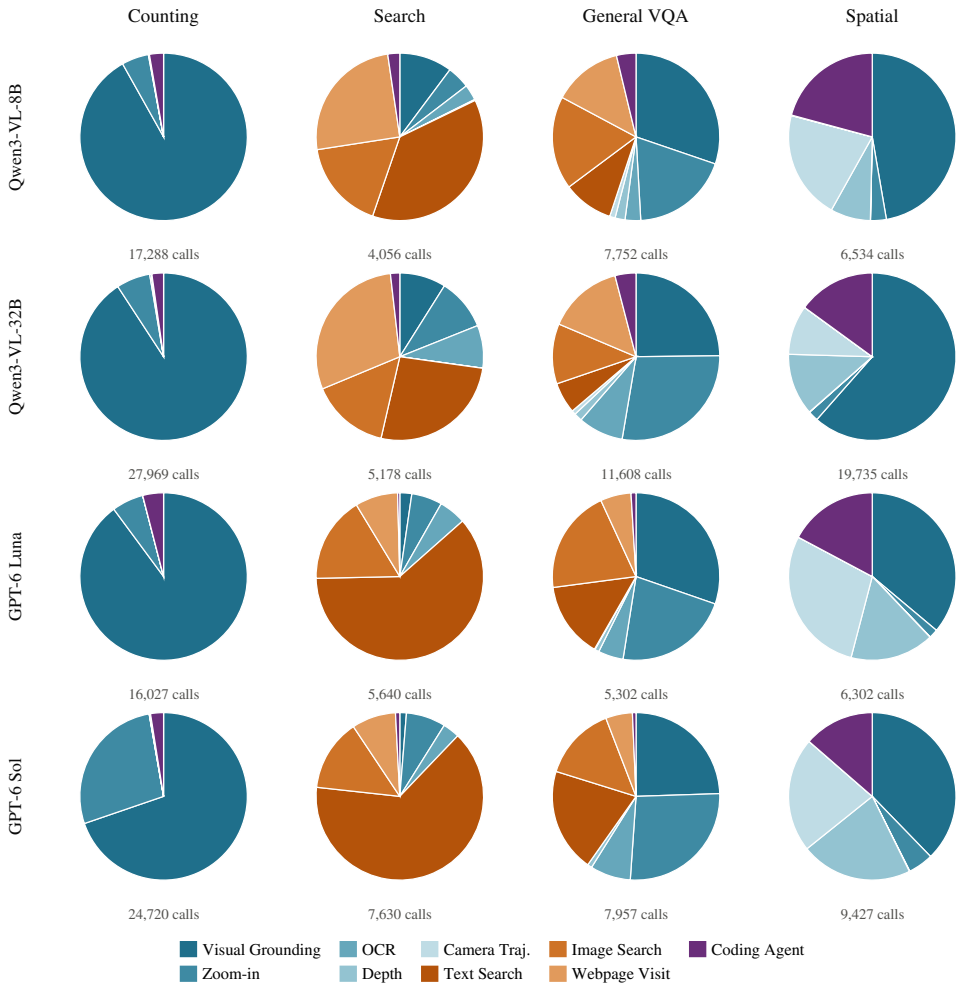


Figure 7: **Distribution of tool calls by domain.** Calls issued in each domain as a share of that domain's calls, with the call total below each panel. Colors are consistent across panels. Counting and grounding is dominated by Visual Grounding for every model except GPT-6 Sol; search is the only domain dominated by the web capabilities; spatial reasoning is the only domain in which four capabilities carry comparable load.

A.3 Effect of Tool Use on Accuracy

Model	Rows	Base	OpenVLHarness	Δ	Rescued	Broken	Calls per net gain
Qwen3-VL-8B	24,944	50.54	60.64	+10.10	4,745	2,000	12.7
Qwen3-VL-32B	24,927	54.28	62.74	+8.46	4,464	2,181	27.7
GPT-6 Luna	24,230	54.91	65.31	+10.40	4,091	1,387	11.6
GPT-6 Sol	24,234	64.81	73.05	+8.24	3,178	1,045	23.3

Table 4: **OpenVLHarness against the base model, paired per question.** Rescued counts questions answered incorrectly by the base model and correctly by OpenVLHarness; broken counts the converse. Calls per net gain divides the total number of tool calls issued on these questions by the net number of questions whose outcome changed.

Aggregate effect. All four models improve under OpenVLHarness, by $+8.2$ (GPT-6 Sol) to $+10.4$ (GPT-6 Luna) points. The net figure is the balance of two considerably larger flows: between $3{,}178$ and $4{,}745$ questions per model are corrected, while between $1{,}045$ and $2{,}181$ are degraded. Normalizing by the tool calls spent, the cost of a net corrected question ranges from 11.6 calls (GPT-6 Luna) to 27.7 calls (Qwen3-VL-32B), the latter combining the highest call volume with one of the smaller gains.

Null control. On questions where the agent issued no tool call, OpenVLHarness reduces to the base model with a longer system prompt, so the expected paired difference is zero. The measured values are $+0.1$, -0.2, -1.5 and -1.8 points across the four models. The proximity to zero supports the interpretation of the remaining differences as effects of tool use rather than of the prompt or the evaluation protocol, and the two negative values indicate a small cost to carrying the tool interface when it is not used, confined to the smaller models.

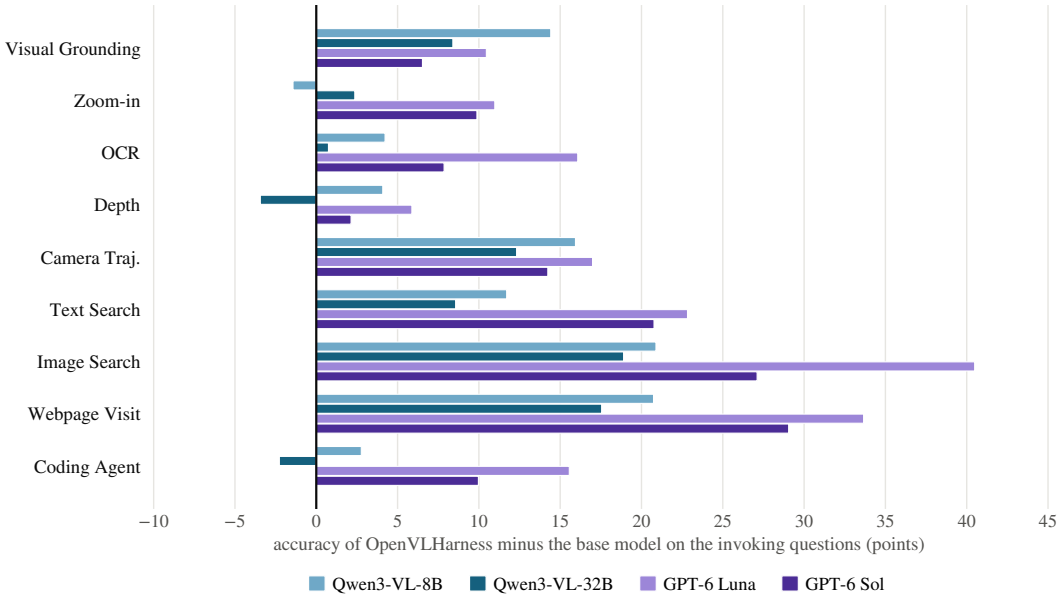


Figure 8: **Per-capability paired comparison against the base model.** For each capability and model, the accuracy of OpenVLHarness minus that of the base model on the questions whose rollout invoked that capability. The corresponding difference on the questions that did not invoke it, which serves as the control, is given in the last column of Table 5 together with the share of questions each capability was invoked on.

Per-capability effects. Figure 8 reports, for each capability and model, the paired difference on the questions that invoked it. Table 5 gives the underlying counts, the share of questions each capability was invoked on, the accuracies of both arms, and the control, that is the same difference

on the questions that did not invoke the capability. The two retrieval capabilities exhibit the largest separation from their controls in every model: rows invoking Image Search improve by 18.9 to 40.5 points, while the corresponding control values lie between -0.6 and $+6.6$. Visual Grounding is the dominant factor for the smaller models — Qwen3-VL-8B improves by 14.4 points on rows that ground and declines by 19.6 points on rows that do not — and flattens for the larger models, which invoke it on nearly all questions. Camera Trajectory is positive for all four models. Depth Estimation, OCR, Zoom-in and the Python Coding Agent are at or below their controls, which is consistent with their being invoked on the questions a model cannot resolve from the image alone rather than with their being ineffective in isolation.

Capability	Model	Questions	Share	Calls	Base	OpenVLHarness	Δ	Δ elsewhere
Visual Grounding	Qwen-8B	13,451	54%	21,657	48.2	62.7	+14.4	−19.6
	Qwen-32B	17,760	71%	40,533	56.7	65.1	+8.4	+0.2
	Luna	13,015	54%	18,376	55.6	66.1	+10.5	−2.2
	Sol	14,895	61%	22,843	68.3	74.8	+6.5	+2.1
Zoom-in	Qwen-8B	1,714	7%	2,727	40.3	38.9	−1.4	+4.8
	Qwen-32B	3,222	13%	5,815	45.2	47.6	+2.4	+7.8
	Luna	2,183	9%	2,455	37.1	48.1	+11.0	+10.2
	Sol	6,087	25%	9,953	48.5	58.3	+9.9	+9.3
OCR	Qwen-8B	312	1%	343	50.6	54.8	+4.2	+7.5
	Qwen-32B	1,259	5%	1,433	60.8	61.5	+0.8	+8.4
	Luna	456	2%	508	35.2	51.3	+16.1	+16.7
	Sol	863	4%	891	62.4	70.2	+7.9	+12.0
Depth	Qwen-8B	598	2%	663	44.1	48.2	+4.1	+4.0
	Qwen-32B	2,187	9%	2,546	59.6	56.2	−3.4	+4.5
	Luna	881	4%	1,064	46.9	52.7	+5.9	+8.0
	Sol	1,854	8%	2,101	63.1	65.2	+2.1	+6.9
Camera Traj.	Qwen-8B	1,444	6%	1,457	34.6	50.6	+15.9	+3.0
	Qwen-32B	1,925	8%	1,975	39.0	51.4	+12.3	+4.2
	Luna	1,733	7%	1,814	44.6	61.6	+17.0	+1.5
	Sol	1,969	8%	2,087	56.0	70.3	+14.2	+3.5
Text Search	Qwen-8B	1,238	5%	2,048	26.1	37.9	+11.7	+8.7
	Qwen-32B	1,353	5%	1,826	34.9	43.5	+8.6	+9.5
	Luna	955	4%	2,884	25.6	48.5	+22.8	+19.6
	Sol	1,801	7%	6,506	39.3	60.1	+20.8	+9.3
Image Search	Qwen-8B	1,885	8%	1,937	32.3	53.2	+20.9	−0.6
	Qwen-32B	1,922	8%	1,974	35.6	54.5	+18.9	+1.3
	Luna	1,662	7%	1,820	19.6	60.0	+40.5	+4.6
	Sol	1,923	8%	2,236	37.6	64.7	+27.1	+6.6
Webpage Visit	Qwen-8B	1,215	5%	1,936	22.1	42.9	+20.7	+7.7
	Qwen-32B	2,177	9%	3,014	34.1	51.7	+17.6	+0.7
	Luna	511	2%	605	13.5	47.2	+33.7	+19.8
	Sol	834	3%	1,079	26.5	55.5	+29.1	+12.2
Coding Agent	Qwen-8B	1,204	5%	2,200	37.0	39.8	+2.8	+8.8
	Qwen-32B	3,319	13%	4,122	57.2	55.0	−2.3	+6.6
	Luna	1,092	5%	1,787	32.6	48.2	+15.6	+13.4
	Sol	906	4%	2,018	37.5	47.4	+10.0	+7.3

Table 5: **Per-capability paired comparison, corresponding to Figure 8.** For each capability and model: the questions whose rollout invoked it at least once, the calls issued on them, the accuracy of the base model and of OpenVLHarness on exactly those questions, their difference, and the same difference computed on the questions that did not invoke the capability. Share is the fraction of paired questions that invoked it, which determines how closely the final column approximates the model's overall gain. A question that invoked three capabilities is counted once under each.

A.4 Capability Composition

Composition across capabilities. Rollouts frequently combine several capabilities rather than invoking one in isolation. Counting the number of *distinct* capabilities per rollout, among engaged rollouts 31.3% (Qwen3-VL-32B) and 26.4% (GPT-6 Sol) invoke two or more, against 20.3% and 16.1% for the smaller models, and the longest rollouts combine four or five. Combinations that cross

capability families are rarer, 5–16% of engaged rollouts, and consist almost entirely of a perception capability followed by the Python Coding Agent: ground or measure first, then compute over the resulting quantities. This is the pattern rewarded by the counting, grounding and spatial datasets, and it is not expressible in a harness that exposes a single capability.

B Additional Results and Analysis

This section reports results with an additional backbone, Kimi K3, gives details of the ablation in Table 2, compares OpenVLHarness with training-based harnesses, analyzes counting and grounding across object counts, extends the search results to 11 information-seeking benchmarks, and tests whether additional reasoning can replace the harness at matched cost.

B.1 Kimi K3 Backbone

Table 6 evaluates OpenVLHarness with GPT-5 (OpenAI, 2025b) and with Kimi K3 (Team et al., 2026), an open-weight mixture-of-experts model with native visual understanding, on the same 23 datasets as Table 1, using the same tools, memory, prompts, and scoring.

		GPT-5		Kimi K3	
Domain	**Dataset**	**Base**	**Ours**	**Base**	**Ours**
Counting and Grounding	FSC-147	22.16	**42.92**	36.78	**42.07**
	CountQA	38.48	**58.70**	71.27	**73.10**
	PixmoCount	76.03	**90.07**	**91.39**	90.45
	OdinW-13	46.27	**63.12**	44.35	**56.91**
	FSCD-147	0.16	**28.09**	7.88	**29.11**
	FSCD-LVIS	2.04	**33.22**	7.15	**33.60**
Visual Search and Deep Research	MMSearch	36.84	**56.14**	67.25	**86.55**
	MMSearch-Plus	10.93	**31.19**	23.47	**55.31**
	RealX-Bench	34.54	**40.72**	51.03	**65.98**
	FVQA	54.67	**55.00**	70.00	**79.33**
	HLE-VL	20.00	**24.85**	41.52	**46.67**
	BC-VL	42.11	**52.38**	51.63	**64.66**
General VQA and Hallucination	RealWorldQA	82.80	**83.13**	83.8	**86.9**
	SimpleVQA	66.33	**68.33**	81.33	**82.33**
	HallusionBench	62.95	**66.07**	69.6	**74.9**
	FREAK-VQA	45.93	**51.44**	52.6	**59.6**
	FREAK-MCQ	44.37	**52.82**	52.9	**63.1**
	V*	72.80	**84.81**	90.6	**92.1**
Spatial Understanding	MMSI-Bench	41.20	**49.80**	46.5	**57.7**
	ERQA	**60.00**	59.00	63.8	**66.8**
	EmbSpatial	81.29	**82.47**	83.7	**83.9**
	SparBench	50.12	**60.09**	60.5	**63.8**
	MindCube	60.09	**80.00**	83.8	**89.6**
	Average	45.74	**57.15**	57.95	**67.15**

Table 6: Results of OpenVLHarness with GPT-5 (OpenAI, 2025b) and Kimi K3 (Kimi Team, 2026) as the backbone, on the same 23 datasets and four task domains as Table 1. Base: the backbone prompted directly; Ours: the backbone with OpenVLHarness.

B.2 Ablation Study Details

The ablation in Table 2 isolates the contribution of each harness component by building a strict superset ladder: every rung adds exactly one mechanism to the one below it, so the gap between adjacent rungs measures that mechanism's marginal value.

Variants. The five rungs are:

1. **Baseline (no tool).** The backbone model prompted directly with the question image and no tool access, establishing the no-harness floor.
2. **+ Tool.** The nine tools are available, but their payloads arrive as raw dumps, their schema is cut to a minimal signature (name, argument names and types, one clause each), and nothing persists between turns: no environment updates, no image labels, no data bindings, no URL references. Tool-generated images are replaced by a short stub.
3. **+ Capability layer.** Adds the capability layer of Sec. 2.1: each tool's raw payload is replaced by a per-tool post-processor that condenses it into a concise, model-readable summary (e.g. grounding boxes on a 0–1000 grid, reranked search results with inline citations, depth/zoom captions), and the full tool schema is restored so the model is told when to reach for each tool and how to phrase its arguments. No memory is present—nothing is carried across turns.
4. **+ Textual memory.** Adds every memory mechanism that is addressed by a *text* handle: per-turn Environment Updates that re-advertise all live state, `webpage_url_N` references that let the agent revisit a URL, and out-of-context data bindings that the Python Coding Agent can read as typed variables. Tool-generated images are replaced by a VLM-written text description (up to 160 tokens) conditioned on the original question. Image references remain absent: tool-produced images cannot be named, stored, or passed to a later tool call.
5. **OpenVLHarness (full).** Adds multimodal image memory: every tool-produced image is labelled (`tool_generated_image_N`), deduplicated by content, and kept addressable. The model selects which image to pass to each tool call by its identifier, enabling compositions such as grounding $\rightarrow$ zoom $\rightarrow$ OCR $\rightarrow$ search.

Implementation. All five variants share the same nine-tool toolset, backbone weights, and decoding configuration (temperature 0.7, top-p 0.8, top-k 20, repetition penalty 1.0, presence penalty 1.5, up to $12{,}800$ generated tokens per turn, 15 tool-calling iterations, 168K-token context cap). They differ only in the *ablation preset* applied to the memory stack: a configuration block that switches individual mechanisms (environment updates, image identifiers, per-tool post-processors, data bindings, URL references) on or off. Each preset is a strict superset of the one below it, validated at initialization so a typo fails the run rather than silently selecting the wrong baseline. Table 7 summarizes the knob settings for each variant.

Variant	Tool schema	Post-proc.	Env update	Bindings	URL refs	Image refs
Baseline	—	—	—	—	—	—
+ Tool	minimal	raw	off	off	off	off
+ Capability layer	full	custom	off	off	off	off
+ Textual memory	full	custom	on	on	on	off
OpenVLHarness	full	custom	on	on	on	on

Table 7: Ablation knob settings for each variant in Table 2. "Post-proc." refers to per-tool output processors (custom vs. raw dumps). "Env update" is the per-turn Environment Update message. "Bindings" are out-of-context typed data bindings. "URL refs" and "Image refs" are symbolic identifiers for web pages and tool-generated images, respectively.

Evaluation setup. Each variant is evaluated on 22 datasets spanning all four domains in Table 1. For efficiency, datasets with more than 500 samples are subsampled to 500 examples with a fixed random seed (42) shared across all variants, so every rung sees exactly the same instances. All reported numbers are means of three independent runs with different sampling seeds. Domain averages in Table 2 are macro-averages over the datasets in each domain.

Domain	Dataset	Training-based AdaReasoner	Training-based PixelReasoner	Training-based DeepEyesV2	Training-free OpenVLHarness
Counting and Grounding	FSC-147	1.34	9.75	21.26	**31.60**
	CountQA	36.98	21.66	21.73	**51.37**
	PixmoCount	77.04	58.44	62.81	**82.35**
	OdinW-13	0.00	**10.24**	8.97	7.80
	FSCD-147	0.00	0.04	0.06	**10.30**
	FSCD-LVIS	0.00	0.00	0.05	**14.20**
	Average	19.23	16.69	19.15	**32.94**
Visual Search and Deep Research	MMSearch	13.45	12.87	18.13	**27.68**
	MMSearch-Plus	0.96	1.93	1.61	**3.86**
	RealX-Bench	11.86	16.49	17.53	**22.51**
	FVQA	21.67	26.33	29.67	**40.33**
	HLE-VL	**10.00**	6.06	3.03	7.27
	BC-VL	18.05	18.30	**23.56**	17.79
	Average	12.66	13.66	15.59	**19.91**
General VQA and Hallucination	RealWorldQA	63.52	**70.33**	66.67	66.14
	SimpleVQA	35.90	41.68	42.12	**46.17**
	HallusionBench	37.43	34.69	18.04	**46.33**
	FREAK-VQA	27.16	35.42	23.90	**36.67**
	FREAK-MCQ	30.08	30.78	26.96	**33.30**
	V*	65.97	**83.01**	81.15	74.87
	Average	43.34	49.32	43.14	**50.58**
Spatial Understanding	MMSI-Bench	20.10	28.10	9.30	**29.80**
	ERQA	26.75	34.25	32.50	**37.00**
	EmbSpatial	**72.01**	71.84	68.93	66.70
	SparBench	31.62	30.33	30.44	**31.90**
	MindCube	31.54	**32.31**	4.80	**32.31**
	Average	36.40	39.37	29.19	**39.54**
Overall average		27.54	29.34	26.66	**35.58**

Table 8: Training-based baselines on the Qwen2.5-VL-7B-Instruct backbone. AdaReasoner, PixelReasoner, and DeepEyesV2 fine-tune the backbone to adapt it to their tool-use harness; OpenVLHarness is training-free and uses the same weights out of the box. Training-based columns are gray; shaded rows are per-domain averages. Bold marks the best result in each row.

B.3 Training-Based Baselines

Table 8 compares OpenVLHarness with the three training-based harnesses on Qwen2.5-VL-7B-Instruct, the base model all three are fine-tuned from (setup in Appendix G.2). Averaged over the 23 datasets, OpenVLHarness reaches 35.58, ahead of PixelReasoner (29.34), AdaReasoner (27.54), and DeepEyesV2 (26.66), without updating any weights.

Where training helps and where it does not. The margin is largest on counting and grounding (32.94 vs. at most 19.23) and on multimodal search (19.91 vs. at most 15.59). The training-based harnesses are optimized for a fixed tool set and answer style: they rarely produce the dense, well-formed box lists that FSCD and OdinW require, and none of them retrieves web evidence in our setup, so knowledge-intensive search questions fall back on the 7B model's parametric knowledge. On general VQA and spatial reasoning, which resemble their training distributions, the gap narrows (50.58 vs. 49.32 and 39.54 vs. 39.37), and the trained models are best on a few perception-centric datasets such as RealWorldQA and V*, where PixelReasoner's learned zoom-in behavior pays off. Training a model into one tool protocol thus helps in-distribution but transfers poorly, whereas a training-free harness can cover all four domains with the same weights.

The backbone as orchestrator. Absolute scores with OpenVLHarness on Qwen2.5-VL-7B are well below those on Qwen3-VL (Table 1). Because the harness relies on the backbone to plan tool calls and to follow output conventions, a weaker orchestrator limits its benefit; on OdinW-13, for example, OpenVLHarness reaches only 7.80 AP with Qwen2.5-VL-7B versus 50.40 with Qwen3-VL-8B. The harness therefore benefits directly from stronger backbones, with no retraining when a new backbone is adopted.

B.4 Counting and Grounding Across Object Counts

Most counting and grounding benchmarks rarely test models at high object counts, which can make headline scores misleading. To make this concrete, we bin every sample of widely used counting and grounding datasets by its number of target objects (Figures 9 and 10). For counting, TallyQA (Acharya et al., 2019), CountBenchQA (Paiss et al., 2023), and PixmoCount place essentially all samples at ten or fewer objects; only FSC-147 and FSCD-LVIS contribute meaningful mass above 50 objects. Grounding is even more skewed: referring-expression datasets such as RefCOCO/+/g (Kazemzadeh et al., 2014; Yu et al., 2016; Mao et al., 2016), Ref-L4 (Chen et al., 2024a), and GroundingME (Li et al., 2025a) ground exactly one target per sample, and detection-style benchmarks such as OdinW-13 and LVIS are dominated by a single object per image (97% and 86% of samples in the 0–1 bucket). Dense, many-object grounding appears only in the FSCD variants. This is why our suite includes FSC-147, FSCD-147, and FSCD-LVIS, and why we break their results down by object count below.

Results by object count. Table 9 breaks down FSC-147 and FSCD-LVIS by the number of ground-truth objects, using the bins defined above. On FSC-147, OpenVLHarness improves exact-match accuracy in every bucket, with the largest gains between 11 and 50 objects (e.g., $5.0 \rightarrow 32.9$ at 8B for 21–50), where the base models' accuracy collapses. On FSCD-LVIS, where every image has at least 21 instances, the base models recover almost none of the instances (F1 ≤ 12.5), whereas OpenVLHarness reaches F1 above 56 up to 100 instances and around 41 beyond. Accuracy nevertheless falls steeply with count for every configuration: above 100 objects exact counting remains essentially unsolved, confirming that the high-count regime is where the most headroom lies.

Additional metrics and benchmarks. Table 10 reports complementary counting metrics and an additional grounding benchmark. OpenVLHarness lowers the mean absolute error on FSC-147 and PixmoCount and raises CountQA relative accuracy, showing that it also reduces large miscounts rather than only fixing near misses. On LVIS (federated val), which queries both present and absent categories over 1,203 classes, OpenVLHarness more than doubles AP at both scales ($18.1 \rightarrow 45.2$ at 8B and $22.9 \rightarrow 45.5$ at 32B).

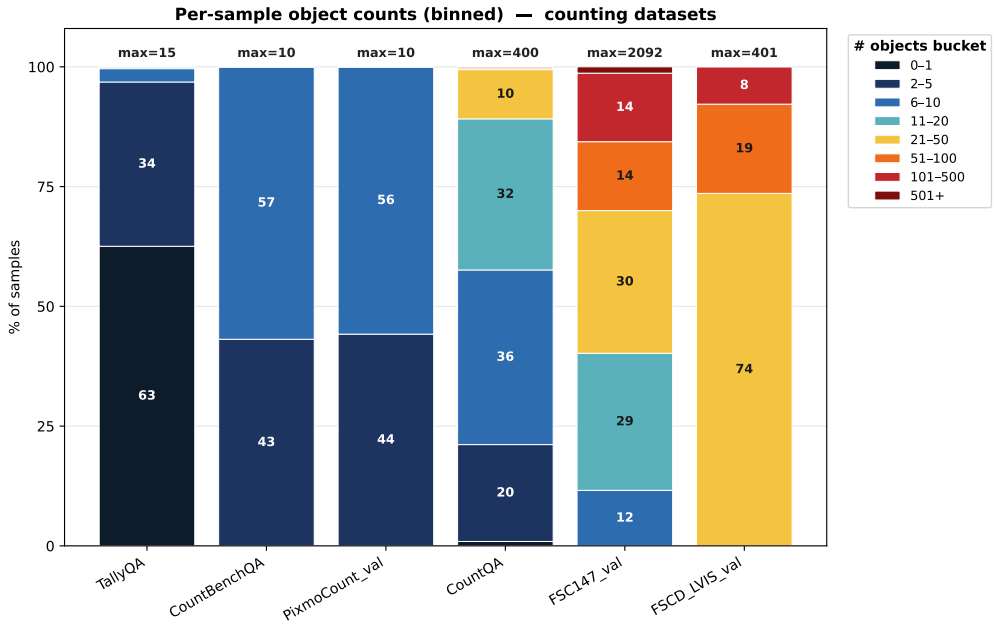


Figure 9: **Object-count distribution across counting benchmarks.** Each bar shows the percentage of samples falling into binned object-count buckets, with the per-dataset maximum count annotated above. Widely used benchmarks such as TallyQA, CountBenchQA, and PixmoCount are dominated by low counts: nearly all of their samples contain ten or fewer objects (and a majority of TallyQA fall in the 0–1 bucket), capping at 10–15. Only FSC147 and FSCD-LVIS contain substantial mass in the high-count regime (> 50 objects), reaching into the hundreds or thousands. This skew means accuracy on standard counting benchmarks largely reflects performance on small object counts and reveals little about high-count visual measurement.

		Qwen3-VL-8B		Qwen3-VL-32B	
Objects	**# Samples**	Base	OpenVLHarness	Base	OpenVLHarness
FSC-147: exact-match accuracy (%)					
≤ 10	149	42.3	**73.8**	64.4	**81.9**
11–20	368	28.8	**59.2**	33.4	**63.3**
21–50	383	5.0	**32.9**	5.2	**34.5**
51–100	185	1.1	**9.7**	2.2	**9.7**
> 100	201	3.5	**6.5**	0.5	**1.0**
FSCD-LVIS: box F1 at IoU 0.5 (%)					
21–50	869	8.5	**63.2**	12.5	**63.5**
51–100	220	3.4	**58.0**	5.8	**56.9**
> 100	92	1.0	**41.3**	1.5	**40.7**

Table 9: Counting and grounding performance by number of ground-truth objects. FSCD-LVIS is scored with instance-level F1 (predicted boxes matched to ground truth at IoU ≥ 0.5) because AP is not defined per sample.

		FSC-147	PixmoCount	CountQA		LVIS
Model	**Method**	MAE ↓	MAE ↓	RA@5 ↑	RA@10 ↑	AP ↑
Qwen3-VL-8B	Base	29.70	0.31	33.64	43.13	18.12
	OpenVLHarness	**23.17**	**0.18**	**50.00**	**57.59**	**45.19**
Qwen3-VL-32B	Base	60.70	0.25	46.79	58.90	22.87
	OpenVLHarness	**31.24**	**0.15**	**65.51**	**72.71**	**45.46**

Table 10: Complementary counting metrics and LVIS grounding. MAE is the mean absolute counting error; RA@k is the share of predictions within $k\%$ of the ground truth.

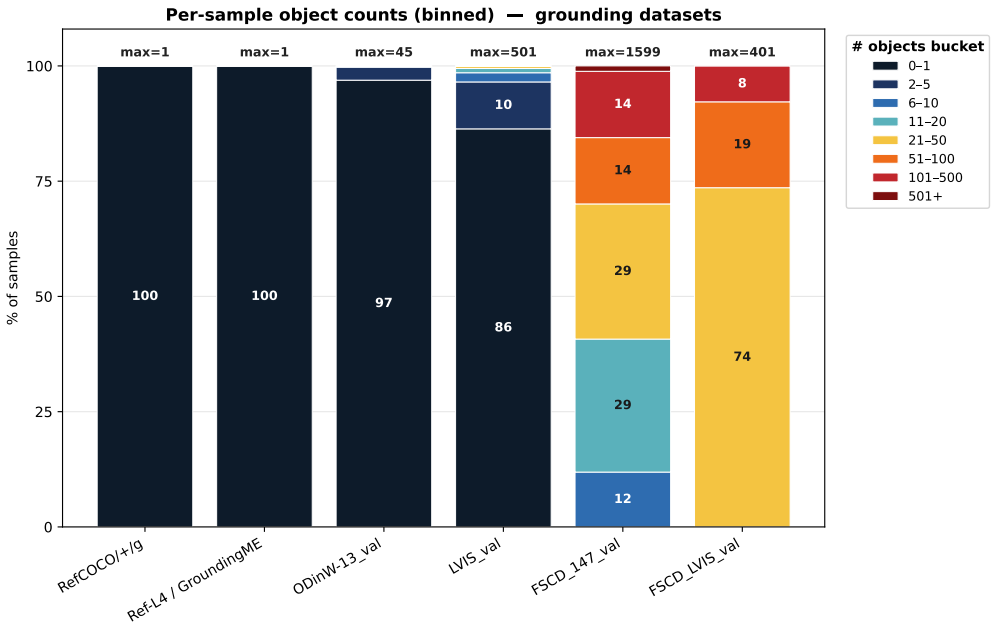


Figure 10: **Object-count distribution across grounding benchmarks.** Each bar shows the percentage of samples falling into binned object-count buckets, with the per-dataset maximum count annotated above. Referring- expression benchmarks such as RefCOCO/+/g and Ref-L4/GroundingME ground a single target per sample (100% in the 0–1 bucket, max $= 1$), and even detection-style sets like ODinW-13 (97%) and LVIS (86%) are overwhelmingly concentrated at one object per image. Only the FSCD variants exercise dense, many-object grounding. As with counting, this distribution implies that progress on common grounding benchmarks is measured almost entirely in the single- or few-object regime.

B.5 Extended Multimodal Search Results

Table 11 extends the search results to 11 information-seeking benchmarks, including five that are not in Table 1. The agent workflow improves every backbone: the average across the 11 benchmarks rises from 13.74 to 29.05 for Qwen3-VL-8B, from 19.21 to 34.61 for Qwen3-VL-32B, and from 31.58 to 41.80 for GPT-5. Gains are largest for the open models, whose parametric knowledge is most limited; GPT-5's direct-inference accuracy is additionally depressed by refusals on identity queries (Appendix D.2). With the harness, Qwen3-VL-32B surpasses direct GPT-5 on average (34.61 vs. 31.58).

Model	MMS	MSP	RXB	FVQA	HLE-VL	BC-VL	MM-BC	VDR	VBB	InfoSeek	Enc-VQA	*Avg*
Direct Inference												
Qwen3-VL-8B	14.04	1.29	22.16	28.39	6.36	21.05	4.94	5.40	7.69	29.20	10.61	13.74
Qwen3-VL-8B-T	12.87	3.22	22.16	25.67	10.30	33.83	7.41	8.20	11.24	27.25	18.06	16.38
Qwen3-VL-32B	20.47	2.57	24.74	32.67	10.00	28.82	6.58	9.40	10.06	33.20	32.76	19.21
GPT-5	36.84	10.93	34.54	54.67	20.00	42.11	8.64	14.20	28.40	53.67	43.33	31.58
Agent Workflow												
Qwen3-VL-8B	51.46	14.79	34.71	55.00	10.30	32.33	6.00	14.40	11.24	50.67	38.67	29.05
Qwen3-VL-32B	62.57	17.68	40.21	62.67	11.52	33.08	10.29	18.80	19.53	55.33	49.00	34.61
GPT-5	56.14	31.19	40.72	55.00	24.85	52.38	25.93	23.80	49.11	52.00	48.67	41.80

Table 11: Extended multimodal search results on 11 information-seeking benchmarks, five of which (MM-BC, VDR, VBB, InfoSeek, Enc-VQA) are not in Table 1. Qwen3-VL-8B refers to Qwen3-VL-8B-Instruct and Qwen3-VL-8B-T denotes Qwen3-VL-8B-Thinking. Abbreviations: MM-BC = MM-BrowseComp; MMS = MMSearch; MSP = MMSearch-Plus; BC-VL = BrowseComp-VL; RXB = RealX-Bench; VDR = VDR-Bench; VBB = VisBrowse-Bench; Enc-VQA = EncyclopedicVQA. *Avg* is the unweighted mean across the 11 benchmarks; per-benchmark sample counts are in Appendix D.2.

B.6 Can More Reasoning Replace the Harness?

OpenVLHarness answers a question over several model calls, and each call carries tool outputs and memory, so it spends more model tokens per question than prompting the base model directly. This raises the question of whether the harness helps simply by spending more compute: a reasoning model given the same token budget might recover the gain by reasoning for longer. GPT-6 Luna exposes a reasoning-effort setting, which lets us test this directly by raising the Base model's spending until it matches that of OpenVLHarness.

Setup. We use 12 datasets from the four domains of Table 1. As in that table, OpenVLHarness, Base, and Codex run GPT-6 Luna at medium effort. We rerun Base on exactly the same questions at higher efforts (high, xhigh, and max; on V*, max in pro reasoning mode). Cost is the GPT-6 Luna API bill for model tokens: uncached input, cache reads and writes, and output, including hidden reasoning tokens, all priced at Standard-tier rates. It excludes tool execution (e.g., SAM 3, OCR, and search APIs) and judging. For each dataset, the *cost-matched* Base is the lowest tested effort whose cost reaches that of OpenVLHarness; on FSC-147, Base already costs more than OpenVLHarness at medium effort, and on CountQA no tested effort reaches it, so we report the highest (xhigh, $0.93\times$). Scoring follows Table 1. On the search benchmarks, every arm is scored with the GPT-4o judge, as in that table. CountQA uses a fixed 500-question subset (seed 42) shared by all arms. Codex is the Codex CLI agent (OpenAI, 2025a) (Appendix G.1): every question runs in a fresh session with the image attached and web search disabled to prevent model directly search the benchmark golden answer on the internet.

Results. Table 12 reports the results. At matched cost, Base averages 51.53, compared with 49.64 at medium effort and 62.85 for OpenVLHarness, while spending $1.27\times$ as much as OpenVLHarness on average. It remains below OpenVLHarness on every dataset except RealWorldQA (76.34 vs. 76.21). Additional effort helps least where the harness supplies information the model cannot infer. On the three search benchmarks, it changes Base by -1.5 to $+2.7$ points, compared with $+11.0$ to $+26.7$ for OpenVLHarness. On FSC-147 and CountQA, it changes Base by 0.0 and $+0.2$ points, compared with $+12.1$ and $+9.6$. On PixmoCount, V*, MMSI-Bench, and SparBench, it recovers

Domain	Dataset	n	Ours Score	Ours $/1k	Base (medium) Score	Base (medium) Cost	Base (cost-matched) Effort	Base (cost-matched) Score	Base (cost-matched) Cost	Codex Score	Codex Cost	Visual Sketchpad Score	Visual Sketchpad Cost
Counting	FSC-147	1286	**40.82**	0.62	28.69	1.32×	medium	28.69	1.32×	25.74	2.25×	27.84	2.76–8.32×
	CountQA	500	**64.00**	1.82	54.40	0.53×	xhigh‡	54.60	0.93×	51.80	0.80×	51.80	0.53–2.59×
	PixmoCount	534	**89.70**	0.66	78.65	0.46×	max	81.46	1.19×	78.46	1.70×	82.96	0.47–2.48×
Search	RealX-Bench	194	**40.72**	0.79	20.62	0.89×	high	22.68	1.50×	19.59	1.65×	22.68	1.13–3.97×
	FVQA	300	**63.33**	0.68	36.67	0.86×	high	39.33	1.47×	35.67	1.78×	38.00	1.15–3.62×
	BC-VL	399	**50.13**	1.44	39.10	0.69×	high	37.59	1.13×	40.85	1.29×	37.59	0.95–2.61×
General	RealWorldQA	765	76.21	0.58	72.81	0.49×	max	**76.34**	1.39×	72.42	1.89×	75.56	0.61–3.34×
	V*	191	**90.05**	1.04	82.72	0.47×	max†	84.82	1.36×	73.82	1.17×	76.96	0.45–3.50×
	SimpleVQA	2025	**67.70**	0.52	42.67	0.84×	high	44.00	1.46×	42.96	2.05×	46.07	1.00–3.34×
Spatial	MMSI-Bench	1000	**45.30**	1.04	36.30	0.65×	xhigh	39.40	1.17×	36.40	1.41×	37.20	0.58–2.17×
	SparBench	1000	**55.29**	1.55	45.51	0.48×	xhigh	49.20	1.02×	44.32	0.93×	43.66	0.92–3.69×
	MindCube	1040	**70.96**	0.54	57.50	0.87×	high	60.29	1.25×	57.69	2.37×	53.65	2.04–7.97×
Average			**62.85**	—	49.64	0.71×	—	51.53	1.27×	48.31	1.61×	49.50	1.05–3.97×

Table 12: Matching the reasoning cost of direct GPT-6 Luna to OpenVLHarness. All arms use GPT-6 Luna on the same questions. OpenVLHarness, Base (medium), Codex, and Visual Sketchpad use medium reasoning effort as in Table 1; Base (cost-matched) is the lowest tested effort whose cost reaches that of OpenVLHarness. *$/1k* is the model-token cost of OpenVLHarness per 1,000 questions at Standard-tier prices, including reasoning tokens; *Cost* is each arm's model-token cost relative to OpenVLHarness on the same dataset. Tool execution and judging are not included. †Maximum effort in pro reasoning mode. ‡Highest effort tested; no tested effort reached the cost of OpenVLHarness. Visual Sketchpad logged only total input and output tokens per question, so its cost is a range: the lower bound prices all input as cache reads and the upper bound as cache writes, the cheapest and most expensive input rates. CountQA uses a fixed 500-question subset, and search benchmarks are scored with the GPT-4o judge used for Base and OpenVLHarness in Table 1.

25–38% of the gap. Accuracy is not monotonic in effort: at high effort, Base scores below its medium-effort result on CountQA (52.60), PixmoCount (78.28), and V* (80.10).

Codex. Codex spends more on model tokens than OpenVLHarness on 10 of the 12 datasets (1.61× on average) but scores below it on all 12, averaging 48.31, below Base at medium effort (49.64). Placing the model in a general-purpose agent loop therefore does not by itself account for the gains of OpenVLHarness.

Token usage. Table 13 gives the per-question token counts behind these costs. Most input tokens of OpenVLHarness and Codex are cache reads and writes of the context carried across their model calls, whereas Base sends a single prompt. Raising Base's effort leaves its input unchanged and mainly increases its thinking tokens.

Caveats. Costs cover model tokens only; the tool backends used by OpenVLHarness are not priced. For OpenVLHarness on RealX-Bench, FVQA, and BC-VL, the saved usage does not separate cache writes, so their cost is a lower bound. Each arm is a single run.

Dataset	Method	Effort	Score	Input tokens				Output tokens		
				Uncached	Cache read	Cache write	**Total**	Thinking	Other	**Total**
FSC-147	OpenVLHarness	medium	40.82	7	7,770	2,131	9,909	483	74	557
	Base	medium	28.69	268	0	0	268	1,566	25	1,592
	Codex	medium	25.74	3	188	6,754	6,945	1,103	9	1,112
	Visual Sketchpad	medium	27.84	–	–	–	30,131	–	–	2,844
CountQA	OpenVLHarness	medium	64.00	7	13,617	11,978	25,601	318	63	381
	Base	medium	54.40	5,027	263	0	5,290	905	18	923
	Base	high	52.60	4,479	811	0	5,290	1,589	26	1,615
	Base	xhigh	54.60	5,215	75	0	5,290	2,300	34	2,334
	Codex	medium	51.80	3	0	9,360	9,363	563	7	570
	Codex	high	51.40	3	56	9,377	9,436	821	8	828
	Visual Sketchpad	medium	51.80	–	–	–	32,762	–	–	1,264
PixmoCount	OpenVLHarness	medium	89.70	6	7,355	3,968	11,328	134	49	182
	Base	medium	78.65	1,781	0	0	1,781	239	11	250
	Base	high	78.28	1,744	37	0	1,781	344	12	356
	Base	max	81.46	153	68	1,561	1,782	1,149	7	1,156
	Codex	medium	78.46	3	0	8,258	8,261	180	7	187
	Codex	high	79.40	3	17	8,260	8,280	239	7	246
	Visual Sketchpad	medium	82.96	–	–	–	11,600	–	–	385
RealX-Bench	OpenVLHarness	medium	40.72	3,610	11,900	–	15,510	529	84	613
	Base	medium	20.62	1,421	0	0	1,421	1,073	39	1,112
	Base	high	22.68	1,407	15	0	1,421	2,027	52	2,079
	Codex	medium	19.59	3	142	7,569	7,714	669	26	696
	Visual Sketchpad	medium	22.68	–	–	–	19,432	–	–	1,384
FVQA	OpenVLHarness	medium	63.33	2,857	12,874	–	15,730	436	93	529
	Base	medium	36.67	498	0	0	498	1,031	40	1,071
	Base	high	39.33	498	0	0	498	1,843	50	1,893
	Codex	medium	35.67	3	68	6,958	7,029	647	24	671
	Visual Sketchpad	medium	38.00	–	–	–	14,603	–	–	1,272
BC-VL	OpenVLHarness	medium	50.13	5,204	36,316	–	41,520	885	235	1,121
	Base	medium	39.10	1,371	0	0	1,371	1,638	69	1,707
	Base	high	37.59	1,322	49	0	1,371	2,914	83	2,996
	Base	xhigh	40.60	1,291	80	0	1,371	4,331	105	4,437
	Codex	medium	40.85	4	1,918	7,727	9,648	1,704	66	1,769
	Visual Sketchpad	medium	37.59	–	–	–	20,865	–	–	2,314
RealWorldQA	OpenVLHarness	medium	76.21	6	7,915	2,927	10,848	213	60	272
	Base	medium	72.81	1,670	0	0	1,670	220	17	237
	Base	high	73.59	1,670	0	0	1,670	311	17	328
	Base	xhigh	73.99	1,670	0	0	1,670	433	19	452
	Base	max	76.34	47	43	1,581	1,671	1,200	13	1,213
	Codex	medium	72.42	3	0	8,113	8,116	155	13	169
	Codex	high	73.73	3	0	8,113	8,116	208	14	222
	Visual Sketchpad	medium	75.56	–	–	–	13,789	–	–	435
V*	OpenVLHarness	medium	90.05	7	16,048	6,103	22,157	168	68	235
	Base	medium	82.72	4,181	0	0	4,181	133	12	145
	Base	high	80.10	4,137	44	0	4,181	167	13	180
	Base	max†	84.82	4,262	4,612	1,892	10,766	1,386	35	1,421
	Codex	medium	73.82	3	0	9,329	9,332	102	9	111
	Codex	high	76.96	3	0	9,329	9,332	150	9	159
	Visual Sketchpad	medium	76.96	–	–	–	27,640	–	–	389
SimpleVQA	OpenVLHarness	medium	67.70	7	9,229	2,078	11,315	274	69	342
	Base	medium	42.67	541	0	0	541	738	37	775
	Base	high	44.00	538	3	0	541	1,381	44	1,426
	Codex	medium	42.96	3	0	6,937	6,940	395	22	417
	Visual Sketchpad	medium	46.07	–	–	–	10,681	–	–	829
MMSI-Bench	OpenVLHarness	medium	45.30	6	10,574	4,572	15,153	667	63	730
	Base	medium	36.30	3,582	125	0	3,707	619	16	635
	Base	high	38.80	3,449	258	0	3,707	1,060	21	1,082
	Base	xhigh	39.40	3,492	215	0	3,707	1,699	29	1,727
	Codex	medium	36.40	3	0	10,219	10,222	367	9	376
	Visual Sketchpad	medium	37.20	–	–	–	14,430	–	–	923
SparBench	OpenVLHarness	medium	55.29	10	22,070	6,302	28,382	928	178	1,107
	Base	medium	45.51	3,193	0	0	3,193	823	19	841
	Base	xhigh	49.20	3,169	25	0	3,193	2,505	38	2,543
	Codex	medium	44.32	3	0	9,677	9,680	472	9	480
	Visual Sketchpad	medium	43.66	–	–	–	37,542	–	–	2,104
MindCube	OpenVLHarness	medium	70.96	5	5,751	1,995	7,751	408	54	462
	Base	medium	57.50	1,028	350	0	1,378	699	24	723
	Base	high	60.29	1,001	377	0	1,378	1,107	29	1,135
	Codex	medium	57.69	3	0	8,074	8,077	516	16	532
	Visual Sketchpad	medium	53.65	–	–	–	27,744	–	–	1,645

Table 13: Model tokens per question for GPT-6 Luna on the datasets of Table 12, averaged over all questions and summed over every model call made for a question. Input tokens are split into uncached input, cache reads, and cache writes; output tokens into hidden reasoning (thinking) and visible output. Scores follow Table 12. †Pro reasoning mode. For OpenVLHarness on RealX-Bench, FVQA, and BC-VL, only dataset-level averages were recorded, without a cache-write split (–). Visual Sketchpad logged only total input and output tokens per question, so its splits are not available (–).

B.7 CODEX WITH OPEN-SOURCE BACKBONES

Table 1 compares OpenVLHarness with Codex only for GPT-6 Luna. To test whether that comparison depends on the backbone, we run Codex with Qwen3-VL-8B and Qwen3-VL-32B on all 23 benchmarks.

Setup. Codex is the Codex CLI agent (OpenAI, 2025a) in code mode, configured as for GPT-6 Luna (Appendix G.1) except that it also appends a short answer-format instruction to each question. Every question runs in a fresh session with the image attached, web search disabled, and medium reasoning effort; the Qwen3-VL Instruct models have no separate reasoning mode, so the effort setting does not change their decoding. Both models are served with vLLM, and each response is capped at 2,048 tokens, the per-turn limit used for OpenVLHarness. Codex runs model-written code in a read-only sandbox. These results therefore measure the Codex agent loop and prompt without shell-based tools. Counting and grounding use the validation splits, as in Table 1. FSC-147 is scored by the last integer in each answer, and an empty answer counts as wrong.

		Qwen-3-VL-8B			Qwen-3-VL-32B		
Domain	**Dataset**	Base	Codex	OpenVLHarness	Base	Codex	OpenVLHarness
Counting & Grounding	FSC-147	15.32	13.53	**37.71**	18.97	16.41	**39.42**
	CountQA	32.00	29.97	**48.56**	44.70	39.73	**64.14**
	PixmoCount	74.53	68.73	**85.21**	79.59	78.46	**87.64**
	OdinW-13	39.80	33.70	**50.40**	42.39	34.66	**59.13**
	FSCD-147	1.79	0.50	**23.52**	2.88	0.50	**28.45**
	FSCD-LVIS	1.47	0.78	**27.37**	1.79	2.24	**31.63**
Search	MMSearch	14.04	12.28	**51.46**	20.47	17.54	**62.57**
	MMSearch-Plus	1.29	2.57	**14.79**	2.57	7.72	**17.68**
	RealX-Bench	22.16	28.35	**34.71**	24.74	30.41	**40.21**
	FVQA	28.39	33.00	**55.00**	32.67	36.33	**62.67**
	HLE-VL	6.36	5.76	**10.30**	10.00	7.27	**11.52**
	BC-VL	21.05	13.53	**32.33**	28.82	17.29	**33.08**
General VQA	RealWorldQA	71.37	67.58	**72.67**	78.82	76.08	**80.52**
	SimpleVQA	54.33	46.27	**70.00**	59.00	50.12	**70.33**
	HallusionBench	57.76	52.77	**59.07**	61.66	59.80	**63.40**
	FREAK-VQA	**44.68**	34.42	37.05	44.06	43.43	**48.19**
	FREAK-MCQ	36.42	34.51	**36.82**	44.06	**50.10**	45.77
	V*	**83.24**	69.11	**83.24**	85.34	79.06	**88.48**
Spatial	MMSI-Bench	31.60	27.40	**33.00**	35.00	31.30	**38.00**
	ERQA	**44.00**	41.25	43.00	44.75	41.75	**48.00**
	EmbSpatial	78.37	74.51	**78.90**	**80.54**	77.53	80.00
	SparBench	39.55	39.82	**43.73**	46.98	**46.99**	46.76
	MindCube	28.36	29.71	**55.48**	32.40	35.87	**60.19**
Average		35.99	33.05	**47.14**	40.10	38.29	**52.51**

Table 14: Codex with Qwen3-VL backbones on the 23 benchmarks of Table 1. Base and OpenVLHarness are copied from that table. Codex runs the Codex CLI agent in code mode at medium reasoning effort with web search disabled. Bold marks the best result within each model group. Averages are taken over all 23 rows. Each Codex cell is a single run.

Results. Table 14 reports the results. Averaged over the 23 benchmarks, Codex scores below Base with both backbones: 33.05 vs. 35.99 with Qwen3-VL-8B and 38.29 vs. 40.10 with Qwen3-VL-32B, while OpenVLHarness reaches 47.14 and 52.51. Codex beats Base on 5 benchmarks with the 8B model and 7 with the 32B model. The larger gains are on RealX-Bench, FVQA, and MMSearch-Plus, on MindCube ($+1.35$ and $+3.47$ points), and on FREAK-MCQ with the 32B model ($+6.04$); the other gains are below 0.5 points. The largest losses are on V* (-14.13 points with the 8B model), BC-VL (-7.52 and -11.53), SimpleVQA (-8.06 and -8.88), and OdinW-13 (-6.10 and -7.73). OpenVLHarness scores above Codex on every benchmark with the 8B model and on 21 of 23 with the 32B model. The two exceptions are FREAK-MCQ (50.10 vs. 45.77) and SparBench (46.99 vs. 46.76). As with GPT-6 Luna, placing a model in a general-purpose coding-agent loop does not by itself reproduce the gains of OpenVLHarness.

C HARNESS IMPLEMENTATION DETAILS

This section documents the harness of Section 2: the nine capabilities of Section 2.1, how their outputs are kept in multimodal memory, and the agent's system prompt.

Capabilities and backends. Following the generalized capability abstraction of Section 2.1, each capability exposes a fixed interface to the agent: typed arguments in, and a natural-language message plus memory artifacts out. The expert model or service behind the interface is a configurable backend, and switching it changes neither the agent's prompt, the tool schema, nor the memory interface. Table 15 lists the capabilities and their backends; all experiments in this paper use the default backend of each capability, shown in bold.

Capability	Interface	Backends
Search		
Text Search	text query → summarized results with cited URLs	**Serper**; Gemini with Google Search
Image Search	image → visual matches (entities, URLs)	**Serper (Google Lens)**; Gemini with Google Search
Webpage Visit	URLs and goal → goal-relevant evidence	**Serper** with an LLM extractor
Perception		
Zoom-in	image and box → cropped image	**built-in**
Visual Grounding	image and text query → boxes, masks, annotated image	**SAM 3**; Grounding DINO
OCR	image → transcribed text	**PaddleOCR-VL**; DeepSeek-OCR
Depth Estimation	image → depth map	**Depth Anything 3**; VGGT
Camera Trajectory	images → camera poses and trajectory plot	**Depth Anything 3**; VGGT
Compositional computation		
Python Coding Agent	objective → program output and new bindings	**single-shot code generator**; full coding agent

Table 15: The nine capabilities, their interfaces, and their interchangeable backends. Bold marks the backend used in all experiments.

C.1 SEARCH TOOLS

By default, all three search tools route queries through the Serper API[1]; Text Search and Image Search can instead be served by Gemini (Comanici et al., 2025) with Google Search grounding, a costlier but stronger route. The search tools post-process the raw payload with a Qwen3-VL-8B-Instruct model as the VLM summarizer. The summarizer runs at decoding temperature 0 with a 4096-token output cap, and each call retrieves the top 10 raw results before reranking and summarization.

Text Search. Search the web with intelligent query routing and VLM summarization. Given a text query, the tool automatically classifies the query intent and routes it to the most appropriate Serper endpoint, which has general web, news, academic papers, images, maps, local places, shopping, or patents types, and returns a coherent summary of the top-10 raw results with inline citations `[1]`, `[2]`, `...` followed by a citations block of source titles and URLs. This format supports multi-round search: the agent can read the summary for quick understanding and then visit specific cited URLs through Webpage Visit. The summarizer prompt is shown below.

Text Search summarizer prompt (abridged)

You are a research summarizer. Given raw web search results and a user query, do exactly three things:

1. RERANK the numbered organic results by how directly each helps answer the query.
2. KEEP at most K most useful results; emit their original 1-based ranks as `kept_indices`. If none help, emit `kept_indices: []`.

[1] `https://serper.dev/`

3. SUMMARIZE the query-relevant facts from the FULL set of raw results into a concise 3–5 sentence prose paragraph; include named entities, numbers, and dates verbatim.

Knowledge-graph, answer-box, and top-stories blocks are ineligible for `kept_indices` but their facts should still surface in `summary`. Output STRICT JSON with the schema {`kept_indices`, `summary`, `no_results_justification`, `suggested_queries`, `suggested_next_actions`}; when no result is useful, populate the justification and propose alternative queries or downstream tool calls.

Image Search. Identify the objects, landmarks, artwork, text, products, plants, animals, and other entities depicted in an image, using Serper's reverse-image-search (Google Lens) endpoint. Returns a list of visual matches with titles, source URLs, links, and thumbnails. The agent invokes the tool by passing the runtime identifier of the target image. The tool performs reverse search (image $\rightarrow$ information) on a single image per call; finding images from a text description is instead handled by Text Search with an image-search intent. A separate multimodal summarizer prompt (below) reranks the candidates against the query image and produces the same JSON envelope as Text Search.

Image Search summarizer prompt (abridged)

You are a visual-match summarizer. Given raw reverse-image-search results (and, in the multimodal variant, the query image plus each candidate's thumbnail), do exactly three things:

1. RERANK the numbered matches by how likely each identifies the same subject as the query image.
2. KEEP at most K most useful matches as `kept_indices`.
3. SUMMARIZE what the FULL set of raw matches indicates about the query image's likely subject in a concise 3–5 sentence prose paragraph; capture entity names, dates, attributes verbatim.

Output STRICT JSON with {`kept_indices`, `summary`, `no_results_justification`, `suggested_next_actions`}; when no match is useful, justify per-match and propose follow-up actions such as cropping with Visual Grounding before re-searching, or switching to Text Search.

Webpage Visit. Visit one or more webpages, convert them to plain text via Serper APIs, and extract goal-relevant content using a local Qwen3-VL-8B-Instruct model. This tool returns per-URL evidence and a summary. The extraction prompt is shown below.

Webpage Visit extraction prompt

Please process the following webpage content and user goal to extract relevant information:
Webpage Content: {`webpage_content`}
User Goal: {`goal`}
Task Guidelines

1. **Content scanning for rationale**: locate the specific sections directly related to the user's goal.
2. **Key extraction for evidence**: extract the most relevant information; preserve the full original context whenever possible.
3. **Summary output**: organize the result into a concise paragraph with logical flow that connects the evidence back to the goal.

Output: a single JSON object with keys `rational`, `evidence`, and `summary` (each a string; empty string if no content is available). No code fences, no Markdown, no extra keys.

C.2 Perception Tools

Zoom-in. Crop a region of an image given a normalized bounding box on the $[0, 1000]$ scale plus padding, and return the cropped image. Used for closer visual inspection after the agent has already localized a region of interest.

Visual Grounding. Locate and highlight regions of an image that match a text query, backed by SAM 3 (Carion et al., 2025) by default or by Grounding DINO (Liu et al., 2024). It returns an annotated image with matched regions highlighted and numbered, along with bounding boxes (and segmentation masks when the backend provides them).

OCR. Extract readable text from an image, backed by PaddleOCR-VL (PaddlePaddle Team, 2025) by default or by DeepSeek-OCR (Wei et al., 2025a). Returns the transcribed text as a single string. Used when the task is about words, numbers, or printed labels in the image; not for general object localization or segmentation.

Depth Estimation. Estimate a per-pixel depth map for an input image, backed by Depth Anything 3 (Lin et al., 2025) or VGGT (Wang et al., 2025b). Returns a depth image (closer surfaces brighter, farther darker). Used when the task requires reasoning about relative distance, foreground–background separation, or 3D scene structure cues that flat 2D crops cannot resolve.

Camera Trajectory. Given an ordered sequence of at least two images of the same scene, estimate each camera's pose relative to the first image using Depth Anything 3 (Lin et al., 2025) or VGGT (Wang et al., 2025b), and translate the result into a form the model can reason over directly. Rather than returning raw geometry alone, the tool reports, for every image, a plain-language summary of translation and rotation with respect to the reference frame (e.g. "salient direction: backward (-5.43); detailed: left (-0.20), down (+0.28), backward (-5.43)"), the accompanying scalars (camera-centre offset, forward direction, yaw, pitch), an explicit statement of the coordinate convention ($+X$ right, $+Y$ down, $+Z$ forward, with positive yaw turning right), and a bird's-eye-view rendering of the camera positions on the reference frame's X–Z plane. Per-image intrinsics and world-to-camera extrinsics are returned as well for downstream computation. Translations below one meter and rotations below ten degrees are reported as non-salient. Used when a question depends on how the viewpoint moved or rotated across frames, or on multi-view scene structure that no single image conveys; the estimate is metric only up to the model's calibration and requires sufficient scene overlap between the reference image and the other views.

C.3 Python Coding Agent

Python Coding Agent. Run Python over the agent's local computation state and the outputs of previously called tools (bounding boxes, OCR strings, segmentation masks, and so on), returning both standard output and the resulting local variables. The agent describes a computation goal in plain language, and by default a single-shot code-generation session of the same backbone model as the orchestrator writes the snippet using prior tool outputs as named variables before it is executed; the same interface can instead be backed by a full coding agent that inspects and revises its own code before returning. Used for aggregating numbers extracted by OCR, filtering detected regions by bounding-box area, or any cross-tool numerical reasoning the agent cannot complete in its own response.

C.4 Multimodal Memory Artifacts

Table 16 summarizes how each tool's output is retained in the explicit multimodal memory of Section 2.2 and the handle by which the agent re-addresses it on later turns.

Environment Update example. After every tool turn, the harness appends an Environment Update message that re-advertises the live memory namespace. The example below is taken verbatim from a FSCD-147 trajectory after two Visual Grounding calls; when web pages have been retrieved, the message additionally lists the registered `webpage_url_N` references.

Artifact	Source tool	Stored representation in memory	Reusable handle
Image artifacts (returned to the model as a native image attachment + an ID)			
Input image	(question)	Registered verbatim	`input_image_`*n*
Cropped region	Zoom-in	Cropped image, deduplicated by content	`tool_generated_image_`*n*
Grounded / mask overlay	Visual Grounding	Annotated image with numbered regions	`tool_generated_image_`*n*
Colorized depth map	Depth Estimation	Rendered depth visualization	`tool_generated_image_`*n*
Camera-trajectory plot	Camera Trajectory	Bird's-eye-view render of the estimated camera path	`tool_generated_image_`*n*
Retrieved thumbnails	Image / Text Search	Image bytes fetched lazily on first reuse	`tool_generated_image_`*n*
Structured data bindings (kept out of the transcript; exposed as typed, named variables)			
Boxes, scores, labels	Visual Grounding	Pixel/normalized boxes, confidences, labels, image size	named binding (e.g. `grounding_`*query*)
Segmentation masks	Visual Grounding	Per-region masks + scores + counts	named binding
Metric depth array	Depth Estimation	Per-pixel depth (float array) + shape + colormap	named binding
Camera poses	Camera Trajectory	Per-image intrinsic and extrinsic matrices	named binding
OCR results	OCR	Recognized text, per-token boxes, confidences	named binding
Web references (URLs replaced by short tokens in retrieved text)			
Search-result pages	Text / Image Search	URL + domain + provenance	`webpage_url_`*n*
Code outputs	Python Coding Agent	Printed output + any emitted images	text + image handle(s)

Table 16: Tool-generated artifacts retained in the persistent working memory, grouped by how they are represented and re-addressed. Heavy payloads (image bytes, numeric arrays) are stored once and replaced in the transcript by a compact handle; the agent reuses an artifact on a later turn by citing its handle, which the harness resolves back to the underlying data.

Environment Update message (example)

Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1: user_input
- tool_generated_image_1: tool_output:Visual_Grounding_Tool
- tool_generated_image_2: tool_output:Visual_Grounding_Tool
When a tool needs image input, pass one of these IDs via the *_image_id field. Do NOT invent image file names or paths.

Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- segment_finger_food: dict (keys: pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h, orig_img_w)
- segment_small_pastry_with_filling: dict (keys: pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h, orig_img_w)

C.5 System Prompt

Here is the system prompt of the agent.

System Prompt

You are a visual reasoning agent that answers questions about images using a structured Perception-Reasoning-Action framework.
Your goal is to provide accurate, well-reasoned answers. You have access to various tools that compensate for known limitations of vision-language models.
Use native tool-calling for tools. Do not write textual Action/Action Input to call tools.
At most one native tool call is allowed per assistant message/turn.
Use the following format:
Question: the input question you must answer
Observation: your direct perception of the input or the result of the action you took
Thought: you should always think about what to do
Action: the action to take, should be one of the provided tools, or "Final Answer" if you have a final answer and do not need to call any more tools.
Action Input: the input to the action, or the final answer if Action is "Final Answer"
Image-input rule:
- Runtime image IDs are provided in Environment Update messages.
- If a tool needs image input, use those IDs in *_image_path fields.
- Do NOT invent image file names or paths.
Tool-Use Rules
- Call exactly ONE tool per turn. Never include multiple <tool_call> blocks.
- Full tool schemas with parameters and types are provided in the <tools> section below.
- After each tool call, verify whether the result is actually informative for the task.
- If a tool result is empty, missing key values, or non-informative, do NOT guess. Issue another tool call with corrected arguments or use a different tool.
Available Tools
You may call ONE tool per turn to assist with the user query.
You are provided with function signatures within <tools></tools> XML tags:
<tools>
{tools_directory}
</tools>
For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

C.6 Models and Serving

Table 17 lists every model used in our experiments and the role it plays.

Model	Role	Access
Agent backbones		
Qwen2.5-VL-7B-Instruct	Backbone (vs. training-based baselines)	vLLM
Qwen3-VL-8B-Instruct	Backbone	vLLM
Qwen3-VL-32B-Instruct	Backbone	vLLM
Qwen3-VL-8B-Thinking	Direct-inference reference (search)	vLLM
GPT-5	Backbone	OpenAI API
GPT-6 Luna, GPT-6 Sol	Backbones	OpenAI API
Evaluation		
GPT-4o	Judge (search and deep research)	OpenAI API
Qwen3-VL-8B-Instruct	Judge / answer extractor (other domains)	vLLM
Tool backends		
Qwen3-VL-8B-Instruct	Webpage Visit extractor, search summarizer	vLLM
(= orchestrator backbone)	Python Coding Agent	(= backbone serving)
SAM 3	Visual Grounding	local server
PaddleOCR-VL	OCR	local server
Depth Anything 3, VGGT	Depth Estimation, Camera Trajectory	local server

Table 17: Models used in our experiments.

Open-source backbones. All Qwen models (Qwen2.5-VL (Qwen Team, 2025a) and Qwen3-VL (Qwen Team, 2025b)) use the public Hugging Face weights, served in bf16 with vLLM (Kwon et al., 2023) behind an OpenAI-compatible endpoint with native tool calling. We sample at temperature 0.7, top-p 0.8, top-k 20, repetition penalty 1.0, and presence penalty 1.5, with up to $12{,}800$ generated tokens per turn. The agent runs for at most fifteen tool-calling iterations per query, and its context is capped at 168K tokens; when a trajectory approaches the cap, older turns are compacted while the latest Environment Update and the most recent turn are kept verbatim.

Proprietary backbones. We call GPT-5 (OpenAI, 2025b), GPT-6 Luna, and GPT-6 Sol (OpenAI, 2026) through the OpenAI API with native function calling, provider-default sampling, and up to $16{,}000$ output tokens per turn. GPT-5 uses the pinned snapshot `gpt-5-2025-08-07`, and the two GPT-6 models (`gpt-6-luna` and `gpt-6-sol`) use medium reasoning effort. All three use the same tools, multimodal memory, system prompt, and iteration cap as the open-source backbones.

Tool backends. Four tools call a language model internally: Text Search and Image Search use it to rerank and summarize the retrieved results, Webpage Visit uses it to extract goal-relevant content from a fetched page, and the Python Coding Agent uses it to write code for the agent's objective. The three search-related tools use Qwen3-VL-8B-Instruct with greedy decoding, independent of the agent backbone. The Python Coding Agent instead routes to a fresh context of the same backbone model as the orchestrator, so that the code generator's capability scales with the backbone. Every capability uses its default backend (Table 15). Perception tools are backed by SAM 3 (Carion et al., 2025) (Visual Grounding), PaddleOCR-VL (PaddlePaddle Team, 2025) (OCR), and Depth Anything 3 (Lin et al., 2025) or VGGT (Wang et al., 2025b) (Depth Estimation and Camera Trajectory), each served locally; search tools query the Serper API.

Judges. We use GPT-4o (OpenAI, 2024) with the Vision-DeepResearch judge prompt (Appendix D.2) for the multimodal search and deep-research benchmarks, and Qwen3-VL-8B-Instruct for the remaining LLM-judged benchmarks; both judges decode greedily. On the counting benchmarks the judge only extracts the final integer from free-form responses, and correctness is decided by exact match. Grounding benchmarks need no judge and are scored with COCO-style box AP.

D Benchmarks and Evaluation Protocol

Table 18 summarizes the 23 benchmarks of Table 1. The following subsections describe each domain's benchmarks, scoring, and prompts.

Domain	Benchmark	Split	# Samples	Format	Scoring
Counting and grounding	FSC-147	val	1,286	integer	exact match
	CountQA	test	1,528	integer	exact match
	PixmoCount	val	534	integer	exact match
	OdinW-13	val	4,938	labeled boxes	box AP
	FSCD-147	val	1,286	boxes	box AP
	FSCD-LVIS	val	1,181	boxes	box AP
Search and deep research	MMSearch	end-to-end	171	open-ended	GPT-4o judge
	MMSearch-Plus	full	311	open-ended	GPT-4o judge
	RealX-Bench	full	194	open-ended	GPT-4o judge
	FVQA	test	1,800	open-ended	GPT-4o judge
	HLE-VL	full	330	open-ended	GPT-4o judge
	BrowseComp-VL	full	399	open-ended	GPT-4o judge
General VQA and hallucination	RealWorldQA	full	765	multiple choice	option match
	SimpleVQA	full	2,025	open-ended	LLM judge
	HallusionBench	full	951	yes/no	exact match
	FREAK-VQA	full	799	open-ended	LLM judge
	FREAK-MCQ	full	994	multiple choice	option match
	V*	full	191	multiple choice	option match
Spatial understanding	MMSI-Bench	full	1,000	multi-image MC	option match
	ERQA	full	400	multiple choice	option match
	EmbSpatial-Bench	full	3,640	multiple choice	option match
	SparBench	tiny	1,000	MC and numeric	option match / MRA
	MindCube	tiny	1,040	multi-image MC	option match

Table 18: The 23 benchmarks of Table 1. "LLM judge" is Qwen3-VL-8B-Instruct; MRA is mean relative accuracy for numeric answers.

Sampling Parameters. All benchmarks use the same agent configuration. The model is served locally via vLLM (Kwon et al., 2023) and sampled with temperature 0.7, top-p 0.8, top-k 20, repetition penalty 1.0, and presence penalty 1.5, with a cap of $12{,}800$ tokens per model call and a 168K-token context window. The agent uses native tool calls and is allowed at most 15 tool-calling iterations per question; a trajectory that exhausts this budget, or that fails after three API retries, is scored incorrect. Web-page observations are capped at 20 environment updates per trajectory.

Because decoding is stochastic, all Qwen3-VL results are means of three runs with different random seeds; Appendix H describes the protocol and reports standard deviations.

Scoring. Multiple-choice and yes/no answers are matched against the gold option after a rule-based parser extracts the choice from the response; when the parser fails, Qwen3-VL-8B-Instruct extracts the chosen option, without judging correctness. Open-ended answers are graded by an LLM judge: GPT-4o for search and deep research (Appendix D.2) and Qwen3-VL-8B-Instruct for the other domains. Counting and grounding are scored without a judge (Appendix D.1). All methods and backbones on a benchmark share the same prompt and scorer.

D.1 Counting and Grounding

FSC-147. A class-agnostic counting benchmark (Ranjan et al., 2021) covering 147 categories with a point annotation for every object. Scenes are dense (median 27 objects, up to 2,092). We evaluate on the 1,286-image val split and ask "How many *category* are there in the image?". Unlike the original few-shot protocol, we do not provide exemplar boxes, so the task is zero-shot.

CountQA. A benchmark (Tamarapalli et al., 2025) of real-world counting questions designed to stress MLLMs with cluttered, occluded, and densely packed scenes. We use all 1,528 questions of the test split.

PixmoCount. The counting benchmark released with Molmo (Deitke et al., 2025), built from natural web images with 2–10 target objects each. We evaluate on the 534 val images that remain downloadable.

OdinW-13. Thirteen detection datasets from Object Detection in the Wild (Li et al., 2022), spanning aerial and underwater imagery, egocentric hands, thermal images, vehicles, and everyday objects. Each image is posed as one query that lists every category of its source dataset, and the model must return a labeled box for each instance. We use the 4,938 val images and report AP averaged over the thirteen datasets.

FSCD-147 and FSCD-LVIS. The few-shot counting-and-detection benchmarks of Nguyen et al. (2022), which annotate a box for every instance: FSCD-147 re-annotates FSC-147, and FSCD-LVIS draws dense scenes (21–401 instances) from LVIS (Gupta et al., 2019). We use the val splits (1,286 and 1,181 images), query one category per image, and again withhold the exemplar boxes.

LVIS (federated). In addition to the six benchmarks of Table 1, Appendix B.4 reports grounding on the LVIS val set (Gupta et al., 2019) (19,809 images, 1,203 categories) under its federated protocol, which queries both categories present in an image and verified-absent ones, so a model must also learn to return an empty list.

Counting metrics. Counting benchmarks are scored by exact-match accuracy between the predicted and ground-truth integers. A rule-based parser extracts the integer (or number word) from the response; for long free-form responses, Qwen3-VL-8B-Instruct first extracts the final answer with the prompt below, and correctness is still decided by exact match. We additionally report mean absolute error and, for CountQA, relative accuracy within 5% and 10% of the ground truth (Appendix B.4).

Grounding metrics. Grounding benchmarks are scored with COCO box AP averaged over IoU thresholds $0.50{:}0.95$, computed with the official COCO evaluation code. Predicted boxes are parsed from the response (JSON, with a lenient fallback for lists of four numbers) and converted from the 0–1000 grid to pixel coordinates. For OdinW-13, predicted labels are matched to the category names, AP is computed per source dataset, and the thirteen values are averaged. FSCD-147, FSCD-LVIS, and LVIS are evaluated per queried category. No LLM judge is involved.

Task prompts. All methods receive the same task prompts, shown below with category names as placeholders.

Counting prompts

FSC-147. How many {category} are there in the image? Your final answer should be a single integer.

PixmoCount. How many {category} are there in the image? If the same instance appears more than once, it should be counted as multiple instances. Your final answer should be a single integer.

CountQA. {original question} Your final answer should be a single integer.

Grounding prompts

OdinW-13. Locate every instance that belongs to the following categories: '{category$_1$}, {category$_2$}, ...'. Report the bounding boxes in JSON format as a list of objects, one object per detected instance, where each object has a "bbox_2d" key with [x1, y1, x2, y2] and a "label" key with the category name exactly as written above. Coordinates must be normalised on a 0-1000 scale, with (0,0) as the top-left corner and (1000,1000) as the bottom-right corner of the image. [...] If none of the listed categories is present, output an empty list [].

FSCD-147, FSCD-LVIS, and LVIS. Locate all instances of '{category}' in this image. For each instance, output a bounding box as [x1, y1, x2, y2] where coordinates are normalised on a 0-1000 scale (with (0,0) as the top-left corner and (1000,1000) as the bottom-right corner). Your answer should be a list of lists. If there are no instances of '{category}' in the image, output an empty list [] as your answer.

Answer extraction prompt (abridged)

You are an AI assistant who will help me to extract the final answer from a model response to a visual question. You are provided with a question, the ground truth answer, and the model's response. Your task is to extract ONLY the final short answer from the response, matching the format of the ground truth (e.g. if the ground truth is a number, extract a number [...]). Do NOT evaluate whether the answer is correct. Just extract it.
If the response contains reasoning or explanation, ignore it and only return the final answer.
[... three worked examples ...]
Question: {question} Ground truth: {answer} Response: {response} Extracted answer:

D.2 Multimodal Search and Deep Research

FVQA. A fact-based VQA dataset (Wu et al., 2025a) that pairs images with questions whose answers require commonsense or world-knowledge facts not directly observable in the image. Following Vision-DeepResearch (Huang et al., 2026), we evaluate on the official test split of 1800 samples; all samples are search-required.

MMSearch. A multimodal search benchmark (Jiang et al., 2024) of 300 manually curated questions across 14 subfields, designed to assess large multimodal models as multimodal search engines. We evaluate on the End-to-End sub-task, which requires the agent to perform query understanding, retrieval, and answering in a single pipeline. To align with other works for fair comparison, we only use the multimodal subset, which contains 171 samples.

MMSearch-Plus. A harder, provenance-aware extension (Tao et al., 2026) that emphasizes localized visual reasoning, verification under noisy or conflicting retrieval, and multi-step text–image search; each item provides 1–5 images and one or more valid answers. We use the full set of 311 samples for evaluation.

BrowseComp-VL. The vision–language extension of BrowseComp (Wei et al., 2025b) introduced by WebWatcher (Geng et al., 2026), with cross-modal information-seeking questions spanning entertainment, humanities, technology, natural science, and other domains. We use all 399 samples for evaluation.

HLE-VL. The multimodal subset of Humanity's Last Exam (Phan et al., 2025), drawn from frontier-knowledge questions across mathematics, the natural sciences, and the humanities. We use all 330 samples for evaluation.

RealX-Bench. Introduced by DeepEyesV2 (Hong et al., 2026a); expert-verified instances that require integrating fine-grained visual perception, dynamic multi-hop web retrieval, and multi-step logical reasoning. This benchmark contains perception and search types, which make it easy to check the performance improvement coming from visual tools and search tools. We use all 194 samples for evaluation.

Extended benchmarks. The following five benchmarks appear only in the extended search results (Appendix B.5, Table 11). They use the same judge and prompt as the six benchmarks above.

MM-BrowseComp. Hand-crafted multimodal browsing questions (Li et al., 2025b) across several sub-tasks, where the answer-relevant content is hard to find. There are two versions of MM-BrowseComp. We use the newest version, which has 243 multimodal tasks.

VDR-Bench. The benchmark released alongside Vision-DeepResearch (Huang et al., 2026): long-horizon multimodal deep-research instances that interleave many reasoning turns, designed to evaluate sustained multi-step browsing under heavy real-world noise. We use its test-mini split of 500 samples.

VisBrowse-Bench. A benchmark of visual-native search instances (Zhang et al., 2026c) that require cross-validation of multimodal evidence via interleaved text–image retrieval and joint reasoning, including cases where the answer is not present in any text snippet. We use the full 169 samples to evaluate.

InfoSeek. A visual information-seeking benchmark (Chen et al., 2023) of fine-grained questions about entities depicted in images (e.g., species, models, landmarks). The official test split has no public answers, so we evaluate on the 2000-sample test subset used by MMSearch-R1 (Wu et al., 2025a).

EncyclopedicVQA. A benchmark (Mensink et al., 2023) of questions about detailed properties of fine-grained categories. We draw a random subset of 2000 samples from the test split. 40 of its Google Landmarks v2 images are no longer publicly available, which leaves 1960 samples for evaluation.

No retrieval shortcut. Across all multimodal search and deep research benchmarks, we did not observe cases in which the search tools surface the gold answer verbatim from indexed pages. This reduces the likelihood that gains are driven by trivial retrieval of the gold answer.

Judge prompt. To align with recent multimodal deep-research work (Huang et al., 2026), we adopt the same LLM judge and use GPT-4o as the judge model, applying the prompt below uniformly across all search-related benchmarks.

LLM Judge Prompt

You are an impartial judge evaluating whether a deep research report contains the correct answer.
[Question] {question}
[Correct Answer] {correct_answer}
[Deep Research Report] {response}
Task: Determine if the deep research report contains the correct answer anywhere in its content.
Instructions:
1. Read through the entire research report carefully
2. Look for the correct answer anywhere in the report (it may be embedded in paragraphs, tables, or sections)
3. Check if the information in the report is consistent with the correct answer
4. The answer does NOT need to be in a specific format or labelled as "final answer"
5. Provide your reasoning
6. Answer with "yes" if the report contains the correct answer, "no" if it doesn't or contradicts it
Output format: correct: [yes/no] reasoning: [your explanation]

Refusals on Identity Queries. Some of our evaluation benchmarks contain queries that reference a real person, such as "who is the person in this photograph," "what is the affiliation of this researcher". Proprietary models are increasingly trained to decline such queries under their privacy and identification policies, even when the gold answer is a public-figure fact already widely indexed on the open web. We observe this for the GPT models in our setting, which returns a refusal in place of an answer, and the judge will count the refusal as incorrect. Therefore, in some of the benchmarks, the GPT performance is affected to some extent.

D.3 General VQA and Hallucination

This domain tests whether tool use helps, or at least does not hurt, on everyday visual questions and on questions designed to provoke hallucination.

RealWorldQA. Real-world photographs, many taken from vehicles, with multiple-choice questions about spatial layout and scene understanding (xAI, 2024); we use all 765 questions.

SimpleVQA. A factuality benchmark (Cheng et al., 2025) of short, knowledge-bound questions about images. We use all 2,025 questions and grade answers as correct or incorrect with an LLM judge.

HallusionBench. Yes/no questions built from paired original and edited images and charts to expose language hallucination and visual illusion (Guan et al., 2024). Of the 1,129 questions in the released set, we evaluate the 951 that are grounded in an image. Free-form responses are first mapped to yes/no by an LLM extraction pass before scoring. The benchmark defines three accuracies: *aAcc*, the fraction of individual questions answered correctly; *qAcc*, the fraction of question groups for which *every* question is correct; and *fAcc*, the fraction of figures for which every associated question is correct. The latter two are deliberately strict, since a model that answers from language priors tends to succeed on the original image and fail on its edited counterpart. We report the benchmark's *Overall* score, the unweighted mean of the three.

FREAK-VQA and FREAK-MCQ. FREAK (Yin et al., 2026) asks about counter-commonsense images, where answering from priors rather than the pixels leads to hallucination, across detection, counting, attribute, position, and OCR questions. It comes in an open-ended format (799 questions, graded by an LLM judge) and a four-option multiple-choice format (994 questions) that includes "the correct answer is not listed".

V*. Questions about small attributes and relative positions of objects in high-resolution images (Wu & Xie, 2024), where the relevant detail is easy to miss at the model's input resolution; we use all 191 questions.

D.4 Spatial Understanding

The spatial benchmarks are multiple-choice, and several of them provide multiple views of a scene, which exercises the harness's handling of several input images. We follow the evaluation split used by the EASI leaderboard (Cai et al., 2025) and use its official implementation in the VLMEvalKit repository (Duan et al., 2024).

MMSI-Bench. Multi-image spatial-intelligence questions (Yang et al., 2025b) spanning positional relations, attributes, motion, and multi-step reasoning across views; we use the 1,000 questions in the standard (non-circular) setting.

ERQA. The embodied reasoning benchmark released with Gemini Robotics (Team et al., 2025), with 400 multiple-choice questions on spatial reasoning, trajectories, actions, and state estimation, some with several images.

EmbSpatial-Bench. Spatial-relation questions posed from an egocentric viewpoint in embodied 3D scenes (Du et al., 2024); we use all 3,640 questions.

SparBench. Spatial perception and reasoning tasks over single and multiple views (Zhang et al., 2026a). We use the tiny split: 20 tasks $\times$ 50 questions $=$ 1,000 questions. Tasks are scored by type. Eleven multiple-choice tasks (spatial relations, spatial imagination, position matching, camera-motion inference) are scored by option-match accuracy. Eight numeric tasks (depth and distance prediction, object-centric and object-to-object, single- and multi-view) are scored by mean relative accuracy (MRA), which averages the indicator that the relative error falls below a threshold, sweeping the threshold from 0.5 to 0.95 in steps of 0.05, so partially accurate magnitude estimates receive partial credit. One task, view-change inference, uses the benchmark's own view-change metric. We report the *Overall* score: the unweighted mean of the 20 per-task metrics, so each task contributes equally regardless of its scoring type and regardless of the differing scales of accuracy and MRA.

MindCube. Questions that require building a mental model of a scene from a few partial views, covering "among", "around", and rotation settings (Wang et al., 2025c); we use the 1,040-question MindCube-Tiny split.

E Failure Analysis and Harness Optimization Details

This appendix describes how failure diagnosis guides specialization of OpenVLHarness on a user's data. We detail the critique protocol, the mapping from diagnoses to remedy channels, and the two workflows that modify harness text or add task-specific tools without updating model weights. We then provide examples of the resulting edits and tools.

E.1 CRITIQUE PROTOCOL

Failure selection and coverage. We use Kimi K3 to critique failures of the unoptimized OpenVL-Harness with Qwen3-VL-8B, obtaining 9,980 completed critiques across 23 benchmarks. Failures follow each benchmark's judge verdict. For OdinW-13, FSCD-147, and FSCD-LVIS, which report box AP without per-image verdicts, we select images whose predicted boxes have F1 below 0.5 at IoU 0.5. Table 20 reports the completed coverage.

Critic inputs and outputs. Each critique includes the question, reference and predicted answers, full trajectory, input and tool-generated images, and toolbox from the run's configuration. Tool descriptions and argument schemas are supplied with called tools marked, so the critic can assess omissions against the capabilities actually available. For each call, it labels information usefulness (*yes*, *partial*, or *no*) and correctness (*correct*, *wrong*, or *unverifiable*). It also returns one primary failure cause, a proposed fix type and sketch, supporting transcript excerpts, and confidence, all in a structured JSON record. Each critique uses approximately 3k prompt tokens and 0.7–2k completion tokens.

Taxonomy. The critic separately identifies where the trajectory failed and what change could address it (Table 19). A *build-tool* fix requires a new callable, including wrappers that verify, filter, or translate outputs and compositions of existing capabilities. A *text-only* fix changes instructions for using existing tools. These labels are subsequently combined into remedy channels; they do not directly determine which workflow to use.

Label	Meaning
Root cause (exactly one)	
Tool not called	The needed evidence was never gathered, although a tool in the agent's toolbox could have supplied it.
Tool output wrong	A tool was called but returned wrong or missing information (missed or hallucinated boxes, a bad crop, a wrong transcription).
Tool output unusable	A tool returned correct information in a form the model could not use (raw arrays, uninterpreted coordinates, an overload of detections).
Tool output ignored	A tool returned correct, plainly usable information that the model did not incorporate or contradicted.
Reasoning error	The evidence was adequate and used; the error is in reasoning, arithmetic, comparison, or answer format.
Label or question issue	The question is ambiguous or the reference answer looks wrong; the agent's answer is defensible.
Fix type	
Build tool	A new callable is needed: a wrapper around a base tool, a combination of tools, or an output-translation layer.
Text only	Only a tool description or the agent prompt changes.
None	Nothing above would plausibly help.

Table 19: Critique taxonomy. Each failure receives one primary cause and one proposed fix type. Unusable and ignored outputs remain separate during critique and are combined for reporting in Fig. 4 and Table 20.

Human agreement. Human annotators independently labeled 100 randomly sampled critiqued failures, with an average agreement of 92% with the critic. The abridged prompt below specifies the evidence requirements and fixed labels.

Critic prompt (abridged)

You are a strict, evidence-driven critic of a vision-language tool-using agent. You are shown ONE trajectory where the agent answered a visual question wrongly, together with the correct answer and the catalog of tools the agent could call. [...] The single question your critique must answer cleanly is: does fixing this family of failures require NEW TOOL CODE the agent can call, or only a TEXT change? [...]
How to judge each tool call. For every tool the agent called, decide `info_useful` (did the returned information bear on answering this question?) and `info_correct` (was that information actually

right, judged against the image content, the question, and the correct answer?). If a grounding tool reports 3 boxes but the correct answer implies 5 objects, that output is wrong. If you cannot tell, say unverifiable; do not guess.
Root cause. Choose exactly one of `tool_not_called`, `tool_output_wrong`, `tool_output_unusable_form`, `tool_output_ignored`, `reasoning_error`, `label_or_question_issue`. `tool_not_called` must name a tool from the agent's toolbox that would have supplied the missing evidence; if no listed tool could have, describe the missing capability in the fix sketch instead. [...]
Rules. Ground every claim in the transcript: quote the tool output or the agent's text. Never assume a tool was right just because the agent believed it. Do not propose memorizing answers or matching question wording; a fix sketch must describe a general capability or change. Set confidence low when the evidence is thin. Return ONLY a JSON object.

E.2 Remedy Channels and Per-Dataset Results

Each failed trajectory receives two labels from the critic: a *root cause*, describing where it failed, and a *fix type*, indicating whether the proposed remedy requires new tool code, a text change, or neither. We determine its remedy channel jointly from this pair, using the tool-call assessments where needed. For example, an omitted call paired with a build-tool fix maps to tool composition (C), whereas the same cause paired with a text-only fix maps to harness text (Y).

The channels distinguish *adding task-specific tools* (A+B+C) from *refining harness text* (Y). New tools can make evidence interpretation explicit, refine outputs, or compose existing capabilities. Harness text comprises the system prompt, tool descriptions, and evidence-rendering templates; editing it guides when existing tools are called and how their results are used. We first assign cases with fix type *none* or a question/reference-answer issue to Z, then map the remaining cause–fix pairs as follows:

- **A: evidence interpretation tools.** An ignored-output or reasoning-error cause paired with a build-tool fix. We also include text-only fixes for ignored outputs, or for reasoning errors when at least one call returned correct, at least partially useful evidence. These tools turn available evidence into explicit task-level conclusions that the model can more readily use.
- **B: output verification and refinement tools.** A wrong- or unusable-output cause paired with a build-tool fix, motivating systematic verification, filtering, or translation.
- **C: tool composition.** An omitted-call cause paired with a build-tool fix, motivating a task-specific callable that combines existing capabilities into the required operation.
- **Y: harness text for tool-use guidance.** The remaining causes paired with a text-only fix.
- **Z: no proposed remedy.** Cases excluded above because no fix is proposed or the question/reference answer is problematic.

The motivation for A is that correct, useful evidence may still require interpretation before it supports an answer. A reasoning error despite such evidence can therefore indicate difficulty interpreting or incorporating it, as can an ignored output. We include these cases even when the critic proposes a text change, to test whether a tool that makes the relevant conclusion explicit can address the difficulty. This motivates a candidate remedy rather than establishing that interpretation caused every such failure.

Table 20 reports the channel shares among critiqued failures. The final column illustrates why an omitted call alone does not identify the remedy: the critic proposes new tool code for 70% of such failures on SparBench, but only 11% on ERQA. The four datasets in Sec. 3.5 span contrasting profiles: Y accounts for 71% of ERQA and 56% of RealWorldQA failures, while A+B+C account for 75% of SparBench and 62% of FREAK-VQA failures. These shares guide the choice of workflow; they do not predict accuracy gains. Critiques of failures alone cannot establish whether an edit harms previously correct answers or whether the orchestrator will use a generated tool.

Domain	Dataset	#Fail.	Root cause (%)					Remedy channel (%)							
			NC	Wrong	Unus./Ign.	Reas.	Label	A	B	C	Y	Z	Tool	Text	NC→build
Counting & Grounding	FSC-147	803	9	89	1	2	0	2	89	2	7	0	**93**	7	27
	CountQA	751	32	53	4	10	1	12	54	2	31	1	**68**	31	8
	PixmoCount	87	34	59	0	5	2	0	59	2	37	2	**61**	37	7
	OdinW-13	1,016	4	56	2	15	23	12	56	0	9	23	**68**	9	7
	FSCD-147	90	0	79	6	0	16	3	81	0	0	16	**84**	0	0
	FSCD-LVIS	370	0	98	1	0	0	1	98	0	0	0	**100**	0	0
Search	MMSearch	80	49	15	0	21	15	20	14	0	51	15	34	**51**	0
	MMSearch-Plus	280	62	13	4	20	1	19	12	0	67	1	31	**67**	1
	RealX-Bench	125	44	29	4	14	9	10	29	6	47	9	44	**47**	13
	FVQA	753	38	21	5	21	15	21	18	0	46	15	39	**46**	1
	HLE-VL	302	42	14	1	41	3	16	13	8	57	6	37	**57**	20
	BC-VL	265	56	5	3	29	7	25	3	0	66	7	27	**66**	0
General VQA	RealWorldQA	209	50	8	11	32	0	26	13	4	56	0	44	**56**	8
	SimpleVQA	770	14	18	11	42	14	44	18	1	23	15	**62**	23	9
	HallusionBench	258	38	9	4	41	8	15	10	8	57	9	33	**57**	21
	FREAK-VQA	503	32	9	9	49	1	52	10	1	35	2	**62**	35	2
	FREAK-MCQ	628	47	8	5	36	4	36	8	1	50	4	46	**50**	2
	V*	32	38	22	6	34	0	34	25	0	41	0	**59**	41	0
Spatial	MMSI-Bench	660	41	9	6	44	0	36	10	9	44	0	**55**	44	23
	ERQA	228	52	5	5	38	0	16	7	6	71	0	29	**71**	11
	EmbSpatial	765	75	3	4	18	1	6	6	11	76	1	23	**76**	15
	SparBench	542	51	8	10	30	1	24	15	36	24	1	**75**	24	70
	MindCube	463	25	3	2	70	0	40	3	3	54	0	46	**54**	10

Table 20: Per-dataset failure analysis of OpenVLHarness on Qwen3-VL-8B. #Fail.: critiqued failures. Root causes: NC = tool not called, Wrong = tool output wrong, Unus./Ign. = tool output unusable or ignored, Reas. = reasoning error, Label = label or question issue. Remedies: A = evidence interpretation, B = output verification/refinement, C = tool composition, Y = harness text, Z = no proposed remedy. Tool = A+B+C and Text = Y; the larger share is bold. Percentages are rounded independently. NC→build: share of tool-not-called failures the critic marks as needing a new tool. On the box-AP grounding benchmarks, a failure is an image with F1 below 0.5 at IoU 0.5.

E.3 Optimization Algorithms

Both workflows use a coding agent (Codex with GPT-5.5) to inspect trajectories and modify a scratch copy of the harness. Candidates are evaluated by running the Qwen3-VL-8B orchestrator and applying the benchmark judge. Harness search changes instructions and evidence presentation; tool generation adds callable operations built from existing capabilities. Both keep model weights fixed.

Harness search. Algorithm 1 maintains a pool of harness states represented as file overlays on the base repository. The editable components are the system prompt and tool-use policy, tool descriptions, and model-facing output templates. For each parent state, an analysis agent inspects failed trajectories and may test hypotheses on up to 20 tasks with five trials each. A proposer then produces M distinct patches without instance-specific rules. Candidates include individual patches and compatible combinations of up to K patches. Each candidate is evaluated with S samples per instance using paired sampling seeds. The top N parents and children by mean search accuracy survive, with older states winning exact ties. A campaign runbook records proposals and scores for subsequent rounds.

We use N=4, M=4, K=4, S=3, and three rounds. ERQA uses 100 samples and is evaluated on a disjoint 300-sample validation split for assessing leading candidates. Similarly, RealWorldQA uses 100 samples and is reported on the disjoint 665 validation split. The first-round analysis can optionally use an earlier critique report, provided its claims are checked against the current trajectories. Thus, failure diagnosis can guide the choice of workflow without being a required input to the search algorithm.

Tool generation. Algorithm 2 groups source failures into at most D candidate tool directions, each describing a reusable operation and the failures it should address. Supporting examples are divided equally into tuning and held-out subsets. A tool-writing agent implements each candidate by composing existing capabilities; it may invoke tools, execute code, and query the base model. For up to T rounds, the candidate is registered in the scratch harness and evaluated on the tuning subset. A separate diagnosis agent classifies its use as not called, execution failed, evidence inadequate,

Algorithm 1 Harness search

Require: base harness h_0, search split $\mathcal{D}$, optional validation split $\mathcal{V}$, pool size N, proposals per parent M, max combination size K, samples per instance S, rounds R

$\mathcal{P} \leftarrow \{h_0\}$; $\text{score}(h_0) \leftarrow \text{Eval}(h_0, \mathcal{D}, S)$
for $r = 1, \ldots, R$ **do**
  $\mathcal{C} \leftarrow \emptyset$
  for each parent $p \in \mathcal{P}$ **do**
    $a \leftarrow \textsc{Analyze}(p, \text{failures of } p \text{ on } \mathcal{D}, \text{runbook})$ ▷ coding agent; may use the playground
    $\{\delta_1, \ldots, \delta_M\} \leftarrow \textsc{Propose}(p, a)$ ▷ text-only patches to the harness
    for each non-empty $I \subseteq \{1, \ldots, M\}$ with $|I| \leq K$ **do**
      $c \leftarrow p \oplus \{\delta_i\}_{i \in I}$; skip if the patches conflict
      $\text{score}(c) \leftarrow \text{Eval}(c, \mathcal{D}, S)$; $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$
  $\mathcal{P} \leftarrow \text{Top}_N(\mathcal{P} \cup \mathcal{C}, \text{score})$ ▷ older state wins exact ties
  evaluate new members of $\mathcal{P}$ on $\mathcal{V}$ if supplied; BOOKKEEP(runbook, $\mathcal{C}$)
return $\mathcal{P}$

evidence misinterpreted, or used correctly, and the tool-writing agent revises the implementation accordingly. Dataset- or instance-specific answer rules and lookup tables are prohibited.

The final gate requires complete scoring, at least two recovered failures and positive net lift across all supporting examples, and at least one recovered failure with positive net lift on the held-out half, whose trajectories were not shown to the tool-writing agent. Because these examples are selected from baseline failures, this gate measures recovery rather than regressions on initially correct answers. Accepted tools are collected into a bank and evaluated together with the original harness on the target dataset. We use $D \leq 5$, $T = 5$, and 50 examples per direction on SparBench and 30 on FREAK-VQA. For both SparBench and FREAK-VQA, we generate tools on data disjoint from the standard evaluation splits and evaluate the resulting harness on the same splits used in our main experiments.

Algorithm 2 Tool generation

Require: harness h with primitive tools, source failures $\mathcal{F}$, max directions D, forge rounds T

$\{(d_j, \mathcal{S}_j)\}_{j \leq D} \leftarrow \textsc{Analyze}(\mathcal{F})$ ▷ tool directions with supporting samples
$\mathcal{B} \leftarrow \emptyset$
for each direction $(d_j, \mathcal{S}_j)$ **do**
  split $\mathcal{S}_j$ into tuning half $\mathcal{S}_j^{\text{tune}}$ and held-out half $\mathcal{S}_j^{\text{held}}$
  $\tau \leftarrow \textsc{Forge}(d_j, h)$ ▷ coding agent writes a tool composing primitive tools
  for $t = 1, \ldots, T$ **do**
    $\rho \leftarrow \text{Run}(h \cup \{\tau\}, \mathcal{S}_j^{\text{tune}})$; $\kappa \leftarrow \textsc{Diagnose}(\rho)$ ▷ how the agent used τ
    if κ requests an early stop **then break**
    $\tau \leftarrow \textsc{Forge}(d_j, h, \tau, \rho, \kappa)$
  $(\text{fix}, \text{broke}) \leftarrow \text{Lift}(h \cup \{\tau\}, \mathcal{S}_j)$; $(\text{fix}', \text{broke}') \leftarrow \text{Lift}(h \cup \{\tau\}, \mathcal{S}_j^{\text{held}})$
  if all examples scored **and** $\text{fix} \geq 2$ **and** $\text{fix} > \text{broke}$ **and** $\text{fix}' \geq 1$ **and** $\text{fix}' > \text{broke}'$ **then**
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{\tau\}$
return $h \cup \mathcal{B}$

E.4 Optimized Harnesses and Generated Tools

Harness text edits. The RealWorldQA example below combines edits from three search rounds: five system-prompt additions and one addition to the Visual Grounding output template. They guide the orchestrator to inspect small road-scene details, verify counts, and match its visual conclusion to the answer options. The output-template edit illustrates how evidence rendering supports adaptation: it prompts verification of grounded candidates before they are used as counts or relation evidence.

Edits in a top-scoring RealWorldQA harness (additions only)

System prompt, when to use visual tools. For road-scene details such as traffic lights, traffic signs, arrows, lane markings, and whether a driver must stop, use visual tools before answering unless the relevant mark or signal is unmistakably large and clear. Before answering that such a detail is absent, not visible, or has a particular color/direction, ground the likely object or region and inspect the original pixels with zoom if the region is small or distant.
System prompt, using tool results. For count questions with numeric options, compare the filtered candidate count with the option values. If the count is outside the listed values or directly adjacent to a listed value, do one missed-object check using a broader/simple phrase or one targeted zoom of the most uncertain region, then choose the listed option best supported by all evidence.
For counts of printed letters, digits, or symbols, use OCR or a text-focused zoom when available before deciding the count. Unless the question explicitly says "standalone", count occurrences inside longer strings as well as isolated characters.
System prompt, final answer. For multiple-choice questions, before writing `<answer>`, explicitly bind your visual conclusion to the option table. The option letter and option text you output must match each other and match the conclusion in your thinking; if your conclusion is not one of the options, re-check the evidence instead of choosing a nearby option.
If the listed options do not include a none/unknown/cannot-determine choice, never use an unlisted absence answer such as "none of the above", "not visible", or "cannot tell". If your draft conclusion is that the needed cue is absent, do one targeted visual check when a tool can resolve it, then answer with the listed option best supported by the evidence.
Visual Grounding output template (appended after the list of boxes). Treat these as candidate regions, not a verified final count. Before answering count, presence, or relation questions, compare each candidate with the original image and remove duplicates, mirror/reflection copies, partial regions, off-target detections, and objects that fail the question's location or context constraint.

Task-specific tools. Table 21 lists the five accepted tools: two from SparBench failures and three from FREAK-MCQ failures. A fourth FREAK direction, localized text and symbol reading, encountered an execution error before completing the gate. The SparBench tools package depth, camera intrinsics, and cross-view geometry into metric 3D operations, addressing the composition needs of channel C. In the diagnosed SparBench failures, the critic judged 87% of Python Coding Agent calls incorrect, motivating reusable implementations of these computations. The FREAK tools make visibility, layout, and counting decisions explicit, addressing the evidence-interpretation needs of channel A. The visibility verifier recovered the most gate examples among the three FREAK tools; the table reports generation-stage recovery, not a per-tool benchmark ablation.

Together, these examples illustrate two ways to adapt the same general-purpose harness to custom data: refine the instructions and rendered evidence when existing capabilities suffice, or add reusable operations when failures call for explicit interpretation, output refinement, or tool composition.

Tool	What it does	Gate	Held-out
Generated from SparBench failures			
`image_points_to_metric_3d`	Lifts localized image points or box centers into camera coordinates using depth and inverse intrinsics, and returns pairwise distances, distances from the observer, and nearest-point rankings in meters, with a warning when a point lies on a depth edge.	30/50	13/25
`camera_center_image_projector`	Estimates camera poses across ordered views and projects another view's camera center into a reference image, returning its image coordinates and depth together with an annotated image.	40/50	18/25
Generated from FREAK-MCQ failures			
`target_visibility_absence_verifier`	After grounding or zooming, checks whether a small object, part, or marking (a handle, wheel, blade, knob, ...) is actually visible, and returns an explicit visible or not-visible status instead of a raw detection.	12/30	7/15
`object_layout_relation_resolver`	Converts grounded boxes of objects or parts into explicit layout relations: which end or side holds a part, immediate neighbors, row order, left-versus-right counts, and symmetry.	4/30	3/15
`visible_part_count_verifier`	Checks candidate boxes of repeated parts (legs, keys, spokes, ...), merges duplicates, and rejects off-target boxes, returning the visible count rather than the typical one.	2/30	1/15

Table 21: Accepted task-specific tools and recovery on their supporting examples. *Gate*: recovered failures / all examples for that direction. *Held-out*: recovered failures / examples withheld from the tool-writing agent. The held-out subset is included in the gate total. These counts measure recovery on selected failures, not accuracy on the full target benchmark.

F QUALITATIVE EXAMPLES

This section shows seventeen complete OpenVLHarness trajectories, ordered as the datasets appear in Table 1: counting and grounding (Examples 1–3), multimodal search and deep research (Examples 4–9), general VQA (Examples 10–12) and spatial understanding (Examples 13–17). Eleven come from GPT-6 Sol and GPT-6 Luna, taken from the runs behind Table 1; six come from OpenVLHarness runs of Qwen3-VL-32B, whose run directories are given at the foot of each example. Each example starts on a new page under a dark title bar and ends with an "End of Example" rule, and every step header names its example. Each example is laid out the same way:

- **Overview and step notes.** Each example opens with a green overview box that explains in plain words what the task asks, why it is hard without tools, and how the agent solves it. For the proprietary GPT-6 models, each step then opens with a short green "What happens" note that says what the call does and why it matters. We add these notes only for proprietary models, whose reasoning traces we cannot access: the API returns the tool calls and any visible text, but not the model's reasoning, and GPT-6 often calls a tool without writing any text at all. The open Qwen3-VL models expose their reasoning, which they write in `<thinking>` tags before each call and which we show verbatim, so their examples have no step notes. The overviews and notes are our descriptions, written to guide the reader; everything else in the trajectory is verbatim.
- **Inputs, question and results.** The input images and the question exactly as sent to the model, then a table comparing the answer with and without the harness against the ground truth. For counting and grounding the table also shows what SAM 3, the model behind Visual Grounding, returns when queried alone, where the trajectory makes that measurable. For some Qwen3-VL examples no run of the same model without tools is available, and the table says so.
- **Without tools.** The same model's full response to the same question with no tools, where such a run exists.
- **Full trajectory with OpenVLHarness.** Every tool call in order. Each step shows the tool, colour-coded by family (**perception**, **search**, **computation**); the complete arguments the model sent; the text the model wrote alongside the call, when there is any; the tool's output, including the complete code the coding agent wrote and its execution output; and every image the tool returned, which the model sees in its next turn. For grounding examples, whose final answer is a list of coordinates, we add a panel drawing the final boxes on the input image, marked as ours.
- **Harness messages.** On a cream background : the environment updates through which the multimodal memory tells the model what it currently holds (Appendix C.4), including the data bindings that pass tool results to the coding agent.
- **What to notice.** A closing note on how the harness contributed.

All model text, arguments and tool outputs are verbatim, generated directly from the trajectory files. To keep each example readable we shorten in five ways, and every cut is marked in red where it happens. Search results show their first two entries, with the number omitted. Coding-agent execution outputs longer than 70 lines keep their first 25 and last 10 lines; the code itself is always complete. A coding-agent run that failed with an error shows the error but not its code. Other tool outputs longer than 1,500 characters are cut, with the remaining length stated; one such output in Example 11 is a webpage summary that loops, repeating the same sentences for over 128,000 characters. After the first environment update, which is shown in full, each update is summarized as the memory entries it adds, with any new data bindings kept verbatim. We also show escaped newlines inside JSON strings as line breaks, and emoji in search results as [emoji] or [flag]. In Example 5 the question and the model's reasoning are in Chinese; the overview translates the question. The system prompt is given in Appendix C.5.

The examples were chosen by hand to illustrate how the harness composes tools, and are not a random sample. Not every one is a clean win, and the overviews say so where it matters: in Example 3 the no-tool answer fails on coordinate format; in Example 4 the perception step does not contribute; and in Examples 17, 14 and 9 the final answer is correct but only partly supported by the trajectory's evidence.

Example 1 Dataset: FSC-147 Model: GPT-6 Sol

Overview (our summary)

Task. Count the pills in a photograph of about 140 pills of many colours and two shapes, oblong capsules and small round tablets, scattered on a dark surface.
Why it is hard. There are too many to count reliably by eye, and a single detector query does not reliably find both shapes.
How the agent solves it. It runs two grounding queries, one general and one for round pills, then asks the coding agent to merge the two sets of detections without counting any pill twice. The first merge fails silently; the agent notices the impossible result and rewrites the merge, which then finds the five pills the first query missed.

INPUT IMAGES



input_image_1

QUESTION (as sent to the model)

```
Question:  How many pill are there in the image?  Your final answer should be a single
integer.
```

WITHOUT TOOLS (GPT-6 Sol's full response to the same question)

```
137
```

RESULTS

SAM 3 alone (query "pill")	134 ✗
SAM 3 alone (query "round pill")	72 ✗
GPT-6 Sol, no tools	137 ✗
GPT-6 Sol + OpenVLHarness	**139** ✓
Ground truth	139

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 1 · **Step 1** Visual Grounding

WHAT HAPPENS The agent starts with the general query "pill". SAM 3 returns 134 detections, drawn as numbered masks on the image.

ARGUMENTS

```
{
  "query":  "pill",
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 134 objects for the prompt 'pill'.  Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [0, 901, 26, 999], "label":  "pill"},
    {"bbox_2d":  [0, 52, 29, 204], "label":  "pill"},
    {"bbox_2d":  [0, 825, 78, 913], "label":  "pill"},
    {"bbox_2d":  [0, 278, 25, 417], "label":  "pill"},
    {"bbox_2d":  [0, 674, 49, 818], "label":  "pill"},
    {"bbox_2d":  [0, 473, 34, 622], "label":  "pill"},
    {"bbox_2d":  [1, 0, 66, 84], "label":  "pill"},
    {"bbox_2d":  [7, 418, 89, 485], "label":  "pill"},
    {"bbox_2d":  [29, 179, 107, 242], "label":  "pill"},
    {"bbox_2d":  [58, 554, 116, 663], "label":  "pill"},
    {"bbox_2d":  [59, 915, 109, 1000], "label":  "pill"},
    {"bbox_2d":  [65, 0, 106, 53], "label":  "pill"},
    {"bbox_2d":  [90, 307, 135, 375], "label":  "pill"},
    {"bbox_2d":  [97, 202, 156, 287], "label":  "pill"},
    {"bbox_2d":  [102, 4, 149, 75], "label":  "pill"},
    {"bbox_2d":  [103, 603, 159, 688], "label":  "pill"},
    {"bbox_2d":  [106, 380, 189, 520], "label":  "pill"},
    {"bbox_2d":  [107, 762, 157, 889], "label":  "pill"},
    {"bbox_2d":  [122, 59, 198, 204], "label":  "pill"},
    {"bbox_2d":  [128, 960, 184, 1000], "label":  "pill"},
    {"bbox_2d":  [130, 513, 214, 559], "label":  "pill"},
    {"bbox_2d":  [161, 153, 260, 264], "label":  "pill"},
    {"bbox_2d":  [163, 0, 249, 23], "label":  "pill"},
    {"bbox_2d":  [180, 658, 212, 799], "label":  "pill"},
    {"bbox_2d":  [181, 421, 273, 515], "label":  "pill"},
    {"bbox_2d":  [182, 237, 267, 348], "label":  "pill"},
```

[The remaining 5,835 characters of this output are not shown (7,324 in total).]

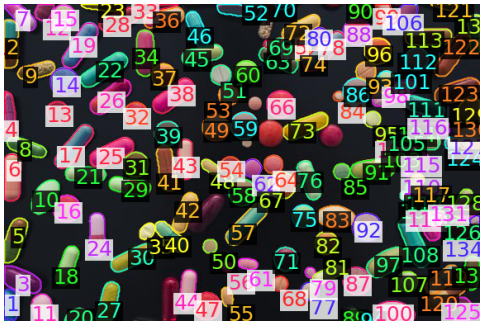


Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_1, from Visual_
Grounding_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- segment_pill:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 1 · **Step 2** Visual Grounding

WHAT HAPPENS A second query, “round pill”, targets the small round tablets that a general query tends to miss. It returns 72 detections, most of which overlap with those of Step 1.

ARGUMENTS

```
{
  "query":  "round pill",
  "image_id":  "input_image_1"
```

```
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 72 objects for the prompt 'round pill'.  Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [91, 307, 136, 374], "label":  "round pill"},
    {"bbox_2d":  [98, 202, 156, 288], "label":  "round pill"},
    {"bbox_2d":  [102, 3, 150, 76], "label":  "round pill"},
    {"bbox_2d":  [104, 603, 160, 688], "label":  "round pill"},
    {"bbox_2d":  [128, 959, 185, 1000], "label":  "round pill"},
    {"bbox_2d":  [250, 467, 303, 547], "label":  "round pill"},
    {"bbox_2d":  [253, 310, 302, 384], "label":  "round pill"},
    {"bbox_2d":  [284, 691, 317, 743], "label":  "round pill"},
    {"bbox_2d":  [311, 190, 358, 262], "label":  "round pill"},
    {"bbox_2d":  [316, 370, 368, 446], "label":  "round pill"},
    {"bbox_2d":  [398, 911, 455, 998], "label":  "round pill"},
    {"bbox_2d":  [415, 491, 450, 538], "label":  "round pill"},
    {"bbox_2d":  [418, 742, 448, 787], "label":  "round pill"},
    {"bbox_2d":  [418, 348, 474, 431], "label":  "round pill"},
    {"bbox_2d":  [434, 192, 476, 258], "label":  "round pill"},
    {"bbox_2d":  [451, 256, 494, 320], "label":  "round pill"},
    {"bbox_2d":  [453, 482, 497, 551], "label":  "round pill"},
    {"bbox_2d":  [456, 893, 480, 956], "label":  "round pill"},
    {"bbox_2d":  [458, 815, 479, 845], "label":  "round pill"},
    {"bbox_2d":  [472, 552, 529, 637], "label":  "round pill"},
    {"bbox_2d":  [476, 342, 533, 427], "label":  "round pill"},
    {"bbox_2d":  [484, 179, 532, 251], "label":  "round pill"},
```

[The remaining 3,087 characters of this output are not shown (4,539 in total).]



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_2, from Visual_
Grounding_Tool).
- segment_round_pill:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_img_
h, orig_img_w)
```

Example 1 · **Step 3** Python Coding Agent

WHAT HAPPENS The agent asks the coding agent to merge the two sets. Both sets are available to the code as data bindings (`segment_pill` and `segment_round_pill`), so no coordinates need to be copied by hand. The run completes but reports zero matches and a union of zero, which cannot be right, since both queries detect the same pills.

ARGUMENTS

```
{
  "objective":  "Match the 134 detected pills against round-pill detections using mask
overlap or bounding box intersection to count round pills absent from original pill
detections, listing locations of missing ones and total union count."
}
```

TOOL OUTPUT

````
Coding agent generated and executed the following Python code:
```python
import numpy as np
from PIL import Image, ImageDraw
import base64
import io
import json

# Decode masks to numpy arrays for intersection computation
def decode_mask(mask_str):
    # Each mask is a base64-encoded string representing a segmentation mask
    # We'll decode it to a numpy array of shape (H, W) with 0/1 values
    # The format is assumed to be a run-length encoded or compressed binary mask
    # For simplicity, we'll treat it as a base64-encoded PNG-like mask
    try:
        # Decode base64 string to bytes
        mask_bytes = base64.b64decode(mask_str)
        # Create a PIL Image from the bytes
        img = Image.open(io.BytesIO(mask_bytes))
        # Convert to numpy array
        mask_array = np.array(img)
        # Ensure binary mask (0 or 1)
        mask_array = (mask_array > 0).astype(np.uint8)
        return mask_array
    except Exception:
        # If decoding fails, return None
        return None

# Compute IoU (Intersection over Union) between two masks
def compute_iou(mask1, mask2):
    if mask1 is None or mask2 is None:
        return 0.0
    # Ensure masks are binary and same shape
    mask1 = (mask1 > 0).astype(np.uint8)
    mask2 = (mask2 > 0).astype(np.uint8)
    # Compute intersection and union
    intersection = np.logical_and(mask1, mask2).sum()
    union = np.logical_or(mask1, mask2).sum()
    if union == 0:
        return 0.0
    return intersection / union

# Compute IoU between two bounding boxes
def compute_bbox_iou(box1, box2):
    # box format:  [x1, y1, x2, y2] (top-left, bottom-right)
    x1_1, y1_1, x2_1, y2_1 = box1
    x1_2, y1_2, x2_2, y2_2 = box2

    # Compute intersection
    x_left = max(x1_1, x1_2)
    y_top = max(y1_1, y1_2)
    x_right = min(x2_1, x2_2)
    y_bottom = min(y2_1, y2_2)

    if x_right <= x_left or y_bottom <= y_top:
        return 0.0

    intersection_area = (x_right - x_left) * (y_bottom - y_top)
    # Compute union
    area1 = (x2_1 - x1_1) * (y2_1 - y1_1)
    area2 = (x2_2 - x1_2) * (y2_2 - y1_2)
    union_area = area1 + area2 - intersection_area

    return intersection_area / union_area if union_area > 0 else 0.0

# Get mask dimensions from original image
H, W = segment_pill["orig_img_h"], segment_pill["orig_img_w"]

# Decode all masks for both datasets
decoded_masks_pill = [decode_mask(mask) for mask in segment_pill["pred_masks"]]
decoded_masks_round_pill = [decode_mask(mask) for mask in segment_round_pill["pred_masks"]]

# Initialize lists to store matched and unmatched round pills
matched_round_pills = []
unmatched_round_pills = []

# For each round pill, find the best match among original pills using IoU
for i, round_mask in enumerate(decoded_masks_round_pill):
    best_iou = 0.0
    best_match_idx = -1
````
````

```
    best_bbox_iou = 0.0
    best_bbox_idx = -1

    for j, pill_mask in enumerate(decoded_masks_pill):
        if pill_mask is not None and round_mask is not None:
            mask_iou = compute_iou(pill_mask, round_mask)
            bbox_iou = compute_bbox_iou(segment_round_pill["pred_boxes"][i], segment_pill["
pred_boxes"][j])
            # Use a combined score or choose the higher one
            combined_score = (mask_iou + bbox_iou) / 2.0
            if combined_score > best_iou:
                best_iou = combined_score
                best_match_idx = j
                best_bbox_iou = bbox_iou
                best_bbox_idx = j

    if best_iou > 0.5:  # Threshold for matching
        matched_round_pills.append((i, best_match_idx, best_iou, best_bbox_iou))
    else:
        unmatched_round_pills.append((i, segment_round_pill["pred_boxes"][i], segment_round_
pill["pred_scores"][i]))

# Count total union of detected pills (unique pills from both sets)
all_masks = decoded_masks_pill + decoded_masks_round_pill
unique_masks = []
for mask in all_masks:
    if mask is not None:
        # Check if mask is already in unique_masks
        if not any(np.array_equal(mask, existing_mask) for existing_mask in unique_masks):
            unique_masks.append(mask)

total_union_count = len(unique_masks)

# Output results
print(f"Total round pills detected: {len(segment_round_pill['pred_masks'])}")
print(f"Total original pills detected: {len(segment_pill['pred_masks'])}")
print(f"Matched round pills: {len(matched_round_pills)}")
print(f"Unmatched round pills: {len(unmatched_round_pills)}")
print(f"Total union count: {total_union_count}")

# List locations of unmatched round pills
print("
Unmatched round pills (location and score):")
for idx, box, score in unmatched_round_pills:
    print(f" Index: {idx}, BBox: {box}, Score: {score}")

# Optional: Print matched pairs
print("
Matched round pills (round_idx, original_idx, IoU, bbox_iou):")
for round_idx, orig_idx, iou, bbox_iou in matched_round_pills:
    print(f" Round {round_idx} -> Original {orig_idx}, IoU: {iou:.4f}, BBox IoU: {bbox_
iou:.4f}")
```

```
Execution output:
Total round pills detected: 72
Total original pills detected: 134
Matched round pills: 0
Unmatched round pills: 72
Total union count: 0

Unmatched round pills (location and score):
  Index: 0, BBox: [52.31292724609375, 117.94264221191406, 78.2960433959961,
143.74490356445312], Score: 0.921875
  Index: 1, BBox: [56.460731506347656, 77.62562561035156, 90.11420440673828,
110.55953979492188], Score: 0.91015625
  Index: 2, BBox: [58.77238082885742, 1.2981762886047363, 86.19441986083984,
29.0463809967041], Score: 0.9140625
  Index: 3, BBox: [59.73981857299805, 231.58026123046875, 91.96317291259766,
264.26812744140625], Score: 0.921875
  Index: 4, BBox: [73.73127746582031, 368.23492431640625, 106.33198547363281,
384.2291259765625], Score: 0.85546875
  Index: 5, BBox: [144.1370849609375, 179.3444061279297, 174.76834106445312,
209.95547485351562], Score: 0.9296875
  Index: 6, BBox: [145.93304443359375, 119.18267822265625, 173.69546508789062,
147.4763641357422], Score: 0.9140625
  Index: 7, BBox: [163.64910888671875, 265.3359680175781, 182.84002685546875,
285.3944396972656], Score: 0.875
```

```
  Index:  8, BBox:  [178.857666015625, 72.8042221069336, 206.23037719726562,
100.41905212402344], Score:  0.93359375
  Index:  9, BBox:  [182.19520568847656, 142.0789031982422, 212.25015258789062,
171.2662811279297], Score:  0.9375
  Index:  10, BBox:  [229.33837890625, 349.857177734375, 262.0947265625,
383.17645263671875], Score:  0.89453125
  Index:  11, BBox:  [238.81997680664062, 188.43115234375, 259.37640380859375,
206.6243896484375], Score:  0.69140625
  Index:  12, BBox:  [240.49752807617188, 284.8372802734375, 258.06903076171875,
302.1544494628906], Score:  0.89453125
  Index:  13, BBox:  [240.52491760253906, 133.4823760986328, 273.088134765625,
165.65768432617188], Score:  0.93359375
  Index:  14, BBox:  [250.11485290527344, 73.80632019042969, 274.3455810546875,
98.89035034179688], Score:  0.93359375
  Index:  15, BBox:  [259.9403076171875, 98.40367126464844, 284.60345458984375,
122.7780990600586], Score:  0.9296875
  Index:  16, BBox:  [260.7716369628906, 185.05271911621094, 286.3943176269531,
211.45742797851562], Score:  0.91015625
```

[47 lines of execution output not shown here (82 in total).]

```
  Index:  64, BBox:  [534.013671875, 160.11773681640625, 550.9052734375,
176.45880126953125], Score:  0.80859375
  Index:  65, BBox:  [535.5726318359375, 355.00115966796875, 566.8264770507812,
384.34881591796875], Score:  0.859375
  Index:  66, BBox:  [538.7619018554688, 260.182861328125, 566.955322265625,
287.40594482421875], Score:  0.89453125
  Index:  67, BBox:  [540.86328125, 147.92442321777344, 552.2364501953125,
159.67457580566406], Score:  0.8203125
  Index:  68, BBox:  [548.8511962890625, 114.78163146972656, 577.1387939453125,
147.60922241210938], Score:  0.890625
  Index:  69, BBox:  [549.4615478515625, 148.23785400390625, 577.478515625,
178.4287872314453], Score:  0.87890625
  Index:  70, BBox:  [555.9362182617188, 242.7733154296875, 575.7222290039062,
259.68731689453125], Score:  0.703125
  Index:  71, BBox:  [563.8709106445312, 256.3594665527344, 577.0241088867188,
274.3338317871094], Score:  0.78125

Matched round pills (round_idx, original_idx, IoU, bbox_iou):
```

Example 1 · **Step 4** Python Coding Agent

WHAT HAPPENS The agent restates the objective more narrowly: compare boxes only, by overlap or centre distance, and do not decode masks. This time the code finds five round detections that match nothing in the first set.

ARGUMENTS

```
{
  "objective":  "Use ONLY the pred_boxes pixel coordinate arrays for the original 134 pill
detections and the 72 round pill detections.  Match round boxes to original boxes by
intersection-over-union >=0.45 or center proximity; report numbered round detections that
are missing from original and original detections with major overlaps (duplicates).  No
mask decoding."
}
```

TOOL OUTPUT

````
Coding agent generated and executed the following Python code:
```python
import numpy as np

# Extract pred_boxes for original pills and round pills
original_boxes = segment_pill["pred_boxes"]
round_boxes = segment_round_pill["pred_boxes"]

# Convert to numpy arrays for easier computation
original_boxes = np.array(original_boxes)
round_boxes = np.array(round_boxes)

# Helper function to compute IoU
def iou(box1, box2):
    # box format:  [x1, y1, x2, y2] (assuming top-left and bottom-right)
    x1_1, y1_1, x2_1, y2_1 = box1
    x1_2, y1_2, x2_2, y2_2 = box2

    # Compute intersection
    x_int = max(0, min(x2_1, x2_2) - max(x1_1, x1_2))
````
````

```
    y_int = max(0, min(y2_1, y2_2) - max(y1_1, y1_2))
    intersection = x_int * y_int

    # Compute union
    area1 = (x2_1 - x1_1) * (y2_1 - y1_1)
    area2 = (x2_2 - x1_2) * (y2_2 - y1_2)
    union = area1 + area2 - intersection

    if union == 0:
        return 0.0
    return intersection / union

# Helper function to compute center proximity (Euclidean distance between centers)
def center_proximity(box1, box2):
    x1_1, y1_1, x2_1, y2_1 = box1
    x1_2, y1_2, x2_2, y2_2 = box2
    center1_x, center1_y = (x1_1 + x2_1) / 2, (y1_1 + y2_1) / 2
    center2_x, center2_y = (x1_2 + x2_2) / 2, (y1_2 + y2_2) / 2
    return np.sqrt((center1_x - center2_x)**2 + (center1_y - center2_y)**2)

# Threshold for matching:  IoU >= 0.45 OR center proximity <= 10 pixels (arbitrary, but
reasonable for visual matching)
MATCH_THRESHOLD_IOU = 0.45
MATCH_THRESHOLD_CENTER = 10.0

# Initialize lists to track matches and unmatched detections
matched_round_indices = set()
matched_original_indices = set()
unmatched_round_indices = []
unmatched_original_indices = []

# For each round pill detection, try to match with original pill detections
for round_idx, round_box in enumerate(round_boxes):
    matched = False
    for orig_idx, orig_box in enumerate(original_boxes):
        iou_val = iou(round_box, orig_box)
        center_dist = center_proximity(round_box, orig_box)

        if iou_val >= MATCH_THRESHOLD_IOU or center_dist <= MATCH_THRESHOLD_CENTER:
            # Found a match
            matched = True
            matched_round_indices.add(round_idx)
            matched_original_indices.add(orig_idx)
            break

    if not matched:
        unmatched_round_indices.append(round_idx)

# Find original detections that have major overlaps (duplicates) - i.e., matched with
multiple round pills
duplicate_original_indices = []
for orig_idx in matched_original_indices:
    count = 0
    for round_idx in range(len(round_boxes)):
        if round_idx not in matched_round_indices:
            continue
        iou_val = iou(original_boxes[orig_idx], round_boxes[round_idx])
        center_dist = center_proximity(original_boxes[orig_idx], round_boxes[round_idx])
        if iou_val >= MATCH_THRESHOLD_IOU or center_dist <= MATCH_THRESHOLD_CENTER:
            count += 1
    if count > 1:
        duplicate_original_indices.append(orig_idx)

# Output results
print("Round detections missing from original:", unmatched_round_indices)
print("Original detections with major overlaps (duplicates):", duplicate_original_indices)
```

```
Execution output:
Round detections missing from original:  [23, 25, 26, 40, 41]
Original detections with major overlaps (duplicates):  []
```

Example 1 · **Final answer**

WHAT HAPPENS The agent adds the five missing pills to the 134 from Step 1 and answers 139, the exact count.

MODEL

```
<answer>139</answer>
```

What to notice. SAM 3 finds 134 pills with the query "pill" and 72 with "round pill"; the two sets overlap, and neither is the answer. The model asks the coding agent to merge them. The first run reports zero matches and a union of zero (Step 3), which cannot be right, since both queries find the same pills. The model rewrites the objective: match by box overlap or centre distance only, with no mask decoding. The second run (Step 4) finds five round pills missing from the first set, and the model answers $134 + 5 = 139$, the exact count. Without tools the model says 137. The detections reach the code as data bindings held in memory, so the model can recover from a bad run by restating the objective instead of re-detecting.

Trajectory:
`outputs/gpt-6-sol/apiagentwithtools/..._cg500_tools/FSC147_val/full_trajectories/1256.json`

END OF EXAMPLE 1

Example 2 Dataset: OdinW-13 (PascalVOC subset) Model: GPT-6 Sol

Overview (our summary)

Task. Detect every object of the twenty PascalVOC categories in a photograph of fourteen men posing around a dinner table, and return labelled boxes.
Why it is hard. The people overlap heavily, some are visible only as a face between two others, and the table is covered with bottles and plates.
How the agent solves it. After a first pass over the four categories present, it zooms on the crowded table and issues narrower queries, ending with a referring expression for one partly hidden man. The final answer matches 22 of the 23 ground-truth objects.

INPUT IMAGES



input_image_1

QUESTION (as sent to the model)

```
Question:  Locate every instance that belongs to the following categories:  'aeroplane,
bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike,
person, pottedplant, sheep, sofa, train, tvmonitor'.  Report the bounding boxes in JSON
format as a list of objects, one object per detected instance, where each object has a "
bbox_2d" key with [x1, y1, x2, y2] and a "label" key with the category name exactly as
written above.  Coordinates must be normalised on a 0-1000 scale, with (0,0) as the
top-left corner and (1000,1000) as the bottom-right corner of the image.  For example:  [{"
bbox_2d":  [x1, y1, x2, y2], "label":  "aeroplane"}, {"bbox_2d":  [x1, y1, x2, y2], "label":
"bicycle"}].  If none of the listed categories is present, output an empty list [].
```

WITHOUT TOOLS (GPT-6 Sol's full response to the same question)


```
[
  {"bbox_2d":[28,205,238,565],"label":"person"},
  {"bbox_2d":[4,421,208,757],"label":"person"},
  {"bbox_2d":[162,413,282,651],"label":"person"},
  {"bbox_2d":[218,397,330,637],"label":"person"},
  {"bbox_2d":[250,269,376,520],"label":"person"},
  {"bbox_2d":[348,232,464,456],"label":"person"},
  {"bbox_2d":[438,216,588,501],"label":"person"},
  {"bbox_2d":[552,243,714,501],"label":"person"},
  {"bbox_2d":[314,381,464,635],"label":"person"},
  {"bbox_2d":[438,408,564,627],"label":"person"},
  {"bbox_2d":[508,387,636,680],"label":"person"},
  {"bbox_2d":[538,432,738,720],"label":"person"},
  {"bbox_2d":[678,451,758,688],"label":"person"},
  {"bbox_2d":[524,379,892,997],"label":"person"},
  {"bbox_2d":[138,632,698,997],"label":"diningtable"},
  {"bbox_2d":[0,741,272,997],"label":"chair"},
  {"bbox_2d":[810,613,908,997],"label":"chair"},
  {"bbox_2d":[264,501,318,643],"label":"bottle"},
  {"bbox_2d":[318,515,374,685],"label":"bottle"},
  {"bbox_2d":[366,560,412,685],"label":"bottle"},
  {"bbox_2d":[428,507,484,677],"label":"bottle"}
]
```

RESULTS (per-image counts from the benchmark evaluator; F1 at IoU ≥ 0.5; ground-truth categories: person, bottle, dining table, chair)

	TP	FP	FN	F1
GPT-6 Sol, no tools	16	5	7	0.73
GPT-6 Sol + OpenVLHarness	**22**	1	1	**0.96**
Ground-truth boxes		23		

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 2 · **Step 1** Visual Grounding

WHAT HAPPENS A first pass on "person" returns 13 people.

ARGUMENTS

```
{
  "query":  "person",
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 13 objects for the prompt 'person'.  Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [0, 422, 191, 871], "label":  "person"},
    {"bbox_2d":  [29, 208, 234, 533], "label":  "person"},
    {"bbox_2d":  [165, 414, 260, 667], "label":  "person"},
    {"bbox_2d":  [211, 402, 324, 636], "label":  "person"},
    {"bbox_2d":  [251, 272, 426, 480], "label":  "person"},
    {"bbox_2d":  [313, 389, 462, 589], "label":  "person"},
    {"bbox_2d":  [381, 236, 478, 505], "label":  "person"},
    {"bbox_2d":  [428, 219, 695, 512], "label":  "person"},
    {"bbox_2d":  [470, 410, 545, 574], "label":  "person"},
    {"bbox_2d":  [521, 384, 635, 642], "label":  "person"},
    {"bbox_2d":  [536, 382, 888, 1000], "label":  "person"},
    {"bbox_2d":  [543, 434, 727, 846], "label":  "person"},
    {"bbox_2d":  [569, 240, 717, 470], "label":  "person"}
]```.  The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_1.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_1, from Visual_
Grounding_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- segment_person:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 2 · **Step 2** Visual Grounding

WHAT HAPPENS Grounding “dining table” returns the table.

ARGUMENTS

```
{
  "query":  "dining table",
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 1 objects for the prompt 'dining table'.  Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [108, 722, 677, 942], "label":  "dining table"}
]```.  The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_2.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_2, from Visual_
Grounding_Tool).
- segment_dining_table:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_
img_h, orig_img_w)
```

Example 2 · **Step 3** Visual Grounding

WHAT HAPPENS Grounding “bottle” returns five bottles on the table.

ARGUMENTS

```
{
  "query":  "bottle",
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 5 objects for the prompt 'bottle'.  Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [271, 495, 327, 642], "label":  "bottle"},
    {"bbox_2d":  [318, 533, 364, 645], "label":  "bottle"},
    {"bbox_2d":  [319, 640, 354, 794], "label":  "bottle"},
    {"bbox_2d":  [369, 576, 409, 705], "label":  "bottle"},
    {"bbox_2d":  [431, 504, 483, 681], "label":  "bottle"}
```

```
]```.  The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_3.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_3, from Visual_
Grounding_Tool).
- segment_bottle:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 2 · **Step 4** Visual Grounding

WHAT HAPPENS Grounding “chair” returns three chairs.

ARGUMENTS

```
{
  "query":  "chair",
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 3 objects for the prompt 'chair'.  Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [0, 745, 260, 999], "label":  "chair"},
    {"bbox_2d":  [0, 485, 61, 635], "label":  "chair"},
    {"bbox_2d":  [785, 612, 939, 999], "label":  "chair"}
]```.  The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_4.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_4, from Visual_
Grounding_Tool).
- segment_chair:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 2 · Step 5 Zoom In

WHAT HAPPENS A zoom on the crowded right half of the table, where people overlap most.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [450, 350, 900, 810],
  "padding":  50
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_GCCULV9QqGvr09iIPPIpWTM2) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_5.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_5, from Zoom_In_Tool).
```

Example 2 · Step 6 Visual Grounding

WHAT HAPPENS The agent re-grounds the table with the shorter query “table”.

ARGUMENTS

```
{
  "query":  "table",
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 1 objects for the prompt 'table'.  Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [109, 741, 677, 940], "label":  "table"}
]```.  The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_6.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_6, from Visual_
Grounding_Tool).
- segment_table:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 2 · **Step 7** Visual Grounding

WHAT HAPPENS The query “man” is an alternative wording for the people, and again returns 13.

ARGUMENTS

```
{
  "query":  "man",
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 13 objects for the prompt 'man'.  Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [0, 422, 190, 870], "label":  "man"},
    {"bbox_2d":  [29, 209, 234, 533], "label":  "man"},
    {"bbox_2d":  [166, 415, 259, 667], "label":  "man"},
    {"bbox_2d":  [212, 402, 324, 636], "label":  "man"},
    {"bbox_2d":  [251, 272, 426, 478], "label":  "man"},
    {"bbox_2d":  [313, 389, 462, 589], "label":  "man"},
    {"bbox_2d":  [381, 236, 478, 504], "label":  "man"},
    {"bbox_2d":  [430, 219, 694, 512], "label":  "man"},
    {"bbox_2d":  [470, 410, 545, 573], "label":  "man"},
    {"bbox_2d":  [521, 384, 636, 642], "label":  "man"},
    {"bbox_2d":  [537, 382, 887, 1000], "label":  "man"},
    {"bbox_2d":  [543, 432, 726, 845], "label":  "man"},
    {"bbox_2d":  [570, 240, 716, 470], "label":  "man"}
]```.  The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_7.
```

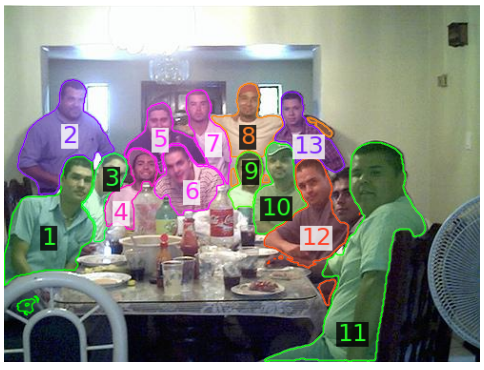


Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_7, from Visual_
Grounding_Tool).
- segment_man: dict (keys: pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h, orig_
img_w)
```

Example 2 · Step 8 — Visual Grounding

WHAT HAPPENS A narrow query for one specific person, “person in dark shirt”, returns a single match.

ARGUMENTS

```
{
  "query": "person in dark shirt",
  "image_id": "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 1 objects for the prompt 'person in dark shirt'. Their
corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: ```json
[
    {"bbox_2d": [251, 273, 426, 483], "label": "person in dark shirt"}
]```. The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_8.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_8, from Visual_
Grounding_Tool).
- segment_person_in_dark_shirt: dict (keys: pred_boxes, pred_scores, pred_masks, num_masks,
orig_img_h, orig_img_w)
```

Example 2 · Step 9 — Zoom In

WHAT HAPPENS A second, tighter zoom on a group of people on the right side of the table.

ARGUMENTS

```
{
  "image_id": "input_image_1",
  "coords": [630, 400, 785, 730],
  "padding": 50
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_vl7F9q9d7GQQotr47vmOtha0) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_9.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_9, from Zoom_In_Tool).
```

Example 2 · **Step 10** **Visual Grounding**

> **WHAT HAPPENS** The last query is a referring expression, “man behind the man in white shirt”, aimed at one partly hidden person. SAM 3 accepts such phrases, and the harness lets the agent use them when a category name is not enough.

ARGUMENTS

```
{
  "query":  "man behind the man in white shirt",
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 7 objects for the prompt 'man behind the man in white shirt'.
Their corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [165, 414, 259, 667], "label":  "man behind the man in white shirt"},
    {"bbox_2d":  [211, 402, 324, 637], "label":  "man behind the man in white shirt"},
    {"bbox_2d":  [381, 235, 478, 506], "label":  "man behind the man in white shirt"},
    {"bbox_2d":  [430, 219, 694, 513], "label":  "man behind the man in white shirt"},
    {"bbox_2d":  [470, 409, 545, 573], "label":  "man behind the man in white shirt"},
    {"bbox_2d":  [522, 384, 635, 642], "label":  "man behind the man in white shirt"},
    {"bbox_2d":  [537, 382, 887, 1000], "label":  "man behind the man in white shirt"}
]```.  The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_10.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_10, from Visual_
Grounding_Tool).
- segment_man_behind_the_man_in_white_sh:  dict (keys:  pred_boxes, pred_scores, pred_masks,
num_masks, orig_img_h, orig_img_w)
```

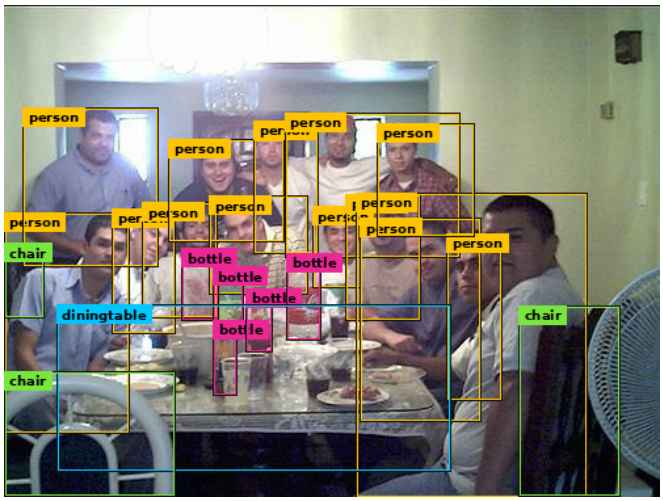
Example 2 · **Final answer**

WHAT HAPPENS The agent answers with 23 labelled boxes; the evaluator matches 22 of them to ground truth, with one false positive and one miss.

MODEL

<answer>[{"bbox_2d":[0,422,191,871],"label":"person"},{"bbox_2d":[29,208,234,533],"label":"person"},{"bbox_2d":[165,414,260,667],"label":"person"},{"bbox_2d":[211,402,324,636],"label":"person"},{"bbox_2d":[251,272,426,480],"label":"person"},{"bbox_2d":[313,389,462,589],"label":"person"},{"bbox_2d":[381,236,478,505],"label":"person"},{"bbox_2d":[428,219,695,512],"label":"person"},{"bbox_2d":[470,410,545,574],"label":"person"},{"bbox_2d":[521,384,635,642],"label":"person"},{"bbox_2d":[543,434,727,846],"label":"person"},{"bbox_2d":[675,465,759,803],"label":"person"},{"bbox_2d":[536,382,888,1000],"label":"person"},{"bbox_2d":[569,240,717,470],"label":"person"},{"bbox_2d":[271,495,327,642],"label":"bottle"},{"bbox_2d":[318,533,364,645],"label":"bottle"},{"bbox_2d":[319,640,354,794],"label":"bottle"},{"bbox_2d":[369,576,409,705],"label":"bottle"},{"bbox_2d":[431,504,483,681],"label":"bottle"},{"bbox_2d":[80,610,680,947],"label":"diningtable"},{"bbox_2d":[0,745,260,999],"label":"chair"},{"bbox_2d":[0,485,61,635],"label":"chair"},{"bbox_2d":[785,612,939,999],"label":"chair"}]</answer>

NOTE Panel drawn by us from the final answer: the 23 predicted boxes on the input image, coloured by label.



What to notice. After grounding the four categories present, the model zooms on the crowded table and issues narrower queries for what the first pass missed: “table”, “man”, “person in dark shirt”, and finally “man behind the man in white shirt” for a partly hidden person. The last query is a referring expression, not a category name; SAM 3 can resolve it, and the harness lets the model use that ability. The final answer matches 22 of the 23 ground-truth objects with one false positive.

Trajectory: `outputs/gpt-6-sol/apiagentwithtools/..._cg500_tools/ODinW13_bbox_val_qwen3/full_trajectories/PascalVOC_val_331.json`

END OF EXAMPLE 2

Example 3 Dataset: FSCD-147 Model: GPT-6 Sol

Overview (our summary)

Task. Box every "package of fresh cut fruit" on a supermarket shelf that holds 53 plastic fruit cups in rows.
Why it is hard. The category name as given is unusual; a detector queried with it finds only part of the shelf.
How the agent solves it. It rephrases the category as "fruit cup", which finds every cup, checks two regions with zooms, and returns 53 boxes that all match. Note that the no-tool answer scores zero only because it used pixel coordinates instead of the requested 0–1000 scale (see the note under the results).

INPUT IMAGES



input_image_1

QUESTION (as sent to the model)

```
Question:  Locate all instances of 'package of fresh cut fruit' in this image.  For each
instance, output a bounding box as [x1, y1, x2, y2] where coordinates are normalised on a
0-1000 scale (with (0,0) as the top-left corner and (1000,1000) as the bottom-right
corner).  Your answer should be a list of lists.  If there are no instances of 'package of
fresh cut fruit' in the image, output an empty list [] as your answer.
```

WITHOUT TOOLS (GPT-6 Sol's full response to the same question)

```
[[0,0,24,24],[24,0,95,24],[98,0,171,24],[173,0,244,24],[247,0,315,24],[319,0,392,24],[395,
0,465,24],[470,0,538,24],[542,0,611,24],[612,0,679,24],[680,0,731,24],[0,88,22,151],[22,88,
101,151],[104,88,175,151],[175,88,246,151],[244,85,319,151],[318,86,391,151],[392,86,464,
151],[463,87,538,151],[539,79,612,149],[611,78,685,149],[687,89,731,149],[0,149,22,204],
[22,149,101,204],[103,149,175,204],[175,149,246,204],[245,149,319,204],[318,149,391,204],
[391,149,464,204],[464,147,538,204],[539,145,612,204],[611,144,685,204],[686,145,731,204],
[0,255,84,341],[84,257,166,341],[165,258,248,341],[247,258,324,341],[326,258,400,341],[400,
258,469,340],[469,258,536,340],[540,258,615,341],[616,258,686,341],[686,260,731,341],[0,
335,84,384],[84,335,166,384],[166,335,248,384],[248,334,324,384],[326,334,400,384],[400,
332,469,384],[469,332,536,384],[540,332,615,384],[616,332,686,384],[686,332,731,384]]
```

RESULTS (F1 at IoU $\geq$ 0.5 against the 53 ground-truth boxes)

	boxes	F1	mean IoU
SAM 3 alone (query "package of fresh cut fruit")	25	0.64 ✗	–
SAM 3 alone (query "fruit cup")	57	0.95 ✗	–
GPT-6 Sol, no tools	53	0.00 ✗	–
GPT-6 Sol + OpenVLHarness	**53**	**1.00** ✓	**0.87**
Ground truth	53		

NOTE The no-tool answer gives its boxes in pixel coordinates (the image is 731×384) instead of the requested 0–1000 scale, so every box misses. Rescaled to that scale, its boxes would score F1 1.00 with mean IoU 0.81. Here the harness's gain is mostly the answer's coordinate frame, plus tighter boxes.

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 3 · **Step 1** Visual Grounding

WHAT HAPPENS Grounding the category exactly as asked returns only 25 candidates, missing much of the shelf.

ARGUMENTS

```
{
  "query":  "package of fresh cut fruit",
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 25 objects for the prompt 'package of fresh cut fruit'.
Their corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [337, 673, 434, 873], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [343, 859, 437, 1000], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [431, 226, 527, 383], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [445, 855, 538, 1000], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [445, 674, 541, 869], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [532, 225, 626, 381], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [535, 381, 625, 526], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [546, 663, 637, 863], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [547, 850, 638, 1000], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [631, 225, 729, 372], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [637, 371, 733, 527], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [640, 669, 731, 859], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [648, 855, 735, 1000], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [738, 208, 829, 372], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [739, 367, 824, 526], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [740, 674, 837, 871], "label":  "package of fresh cut fruit"},
    {"bbox_2d":  [745, 864, 838, 1000], "label":  "package of fresh cut fruit"},
```

[The remaining 745 characters of this output are not shown (2,199 in total).]



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_1, from Visual_
Grounding_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- segment_package_of_fresh_cut_fruit:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_
masks, orig_img_h, orig_img_w)
```

Example 3 · **Step 2** Visual Grounding

WHAT HAPPENS The agent rephrases the query as “fruit cup”. SAM 3 now returns 57 candidates covering the whole shelf, slightly more than the 53 cups.

ARGUMENTS

```
{
  "query":  "fruit cup",
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 57 objects for the prompt 'fruit cup'.  Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [0, 234, 28, 387], "label":  "fruit cup"},
    {"bbox_2d":  [0, 0, 26, 59], "label":  "fruit cup"},
    {"bbox_2d":  [0, 390, 25, 527], "label":  "fruit cup"},
    {"bbox_2d":  [1, 668, 105, 871], "label":  "fruit cup"},
    {"bbox_2d":  [6, 867, 102, 1000], "label":  "fruit cup"},
    {"bbox_2d":  [26, 385, 130, 527], "label":  "fruit cup"},
    {"bbox_2d":  [29, 229, 132, 385], "label":  "fruit cup"},
    {"bbox_2d":  [35, 0, 124, 60], "label":  "fruit cup"},
    {"bbox_2d":  [83, 664, 120, 861], "label":  "fruit cup"},
    {"bbox_2d":  [106, 866, 128, 1000], "label":  "fruit cup"},
    {"bbox_2d":  [114, 676, 212, 867], "label":  "fruit cup"},
    {"bbox_2d":  [120, 865, 212, 1000], "label":  "fruit cup"},
    {"bbox_2d":  [135, 0, 234, 59], "label":  "fruit cup"},
    {"bbox_2d":  [138, 234, 234, 386], "label":  "fruit cup"},
    {"bbox_2d":  [140, 386, 233, 527], "label":  "fruit cup"},
    {"bbox_2d":  [166, 663, 234, 856], "label":  "fruit cup"},
    {"bbox_2d":  [207, 858, 230, 1000], "label":  "fruit cup"},
    {"bbox_2d":  [227, 676, 335, 873], "label":  "fruit cup"},
    {"bbox_2d":  [234, 866, 336, 1000], "label":  "fruit cup"},
    {"bbox_2d":  [235, 0, 333, 57], "label":  "fruit cup"},
    {"bbox_2d":  [236, 230, 331, 387], "label":  "fruit cup"},
    {"bbox_2d":  [236, 386, 331, 526], "label":  "fruit cup"},
    {"bbox_2d":  [332, 385, 432, 526], "label":  "fruit cup"},
```

[The remaining 2,101 characters of this output are not shown (3,568 in total).]



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_2, from Visual_
Grounding_Tool).
- segment_fruit_cup:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 3 · **Step 3** Zoom In

WHAT HAPPENS A zoom on the lower left of the shelf, to check the cups there.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [0, 640, 343, 1000],
  "padding":  50
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_7rYFu8RXWlhnpwbcoWjvJPR2) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_3.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_3, from Zoom_In_Tool).
```

Example 3 · **Step 4** Zoom In

WHAT HAPPENS A zoom on the strip along the top edge, where partly cut-off cups sit.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [0, 0, 1000, 85],
  "padding":  50
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_BAISDAjjqrtvjY61E2BI4mEj) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_4.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_4, from Zoom_In_Tool).
```

Example 3 · Final answer

WHAT HAPPENS The agent returns 53 boxes, and all 53 match a ground-truth cup. The trajectory does not say how the agent chose 53 of the 57 candidates.

MODEL

```
<answer>[[0,0,26,59],[35,0,124,60],[135,0,234,59],[235,0,333,57],[336,0,425,58],[440,0,
521,57],[539,0,619,56],[640,0,723,59],[740,0,829,68],[834,0,921,68],[925,0,1000,68],[0,234,
28,387],[29,229,132,385],[138,234,234,386],[236,230,331,387],[334,220,428,384],[432,226,
527,381],[533,225,626,380],[631,225,729,372],[738,209,830,371],[834,204,927,371],[940,237,
1000,339],[0,390,25,527],[26,385,130,527],[140,386,233,527],[236,386,331,526],[332,385,432,
526],[432,382,528,526],[534,381,625,525],[637,370,733,526],[738,367,823,525],[836,368,924,
526],[937,336,1000,525],[1,668,105,871],[114,676,212,867],[227,676,335,873],[337,673,434,
873],[445,674,541,868],[546,666,637,862],[640,670,731,859],[740,675,836,871],[842,676,936,
872],[939,677,1000,870],[6,867,102,1000],[120,865,212,1000],[234,866,336,1000],[343,859,
437,1000],[445,855,538,1000],[547,851,639,1000],[647,856,735,1000],[745,866,838,1000],[841,
866,934,1000],[936,867,1000,1000]]</answer>
```

NOTE Panel drawn by us from the final answer: the 53 predicted boxes on the input image.



What to notice. The category name as asked, “package of fresh cut fruit”, finds only 25 candidates. The model rephrases it as “fruit cup”, which finds 57, more than the true 53. It zooms on two regions and settles on 53 boxes that match every ground-truth package. The trajectory does not state how the final four candidates were dropped. As the note above says, the no-tool model’s score of zero comes from its coordinate format, not from missing the objects.

Trajectory: `outputs/gpt-6-sol/apiagentwithtools/..._cg500_tools/FSCD_147_bbox_val/full_trajectories/val_950_1.json`

END OF EXAMPLE 3

Example 4 Dataset: MMSearch-Plus Model: GPT-6 Luna

Overview (our summary)

Task. A street photograph from Paris Fashion Week shows a woman in a beige jacket and an orange scarf (faces are blurred in the benchmark image). Which brand's show was she attending, and which event and year was it?
Why it is hard. Nothing in the image names the show; without tools the model guesses a Miu Miu show in 2024.
How the agent solves it. A reverse image search finds no match, so the agent turns the photo into a text description and runs an image search with it. The top result is a street-style gallery titled "What they wore to Hermès" at Paris Fashion Week Fall/Winter 2025, and a page visit confirms the event and season.
Caveat. The perception step contributes nothing: the agent zooms on what it takes for a shopping bag, but the OCR reads a Paris litter-bag slogan ("Ensemble rendons Paris propre"). The answer comes from search alone.

INPUT IMAGES



input_image_1

QUESTION (as sent to the model)

```
Question:  In Paris Fashion Week, what brand did the person in the image represent?  In
which year did the event occur, and what is the event's name?
```

WITHOUT TOOLS (GPT-6 Luna's full response to the same question)

```
The brand is **Miu Miu**.  The event took place in **2024** and was the **Miu Miu Spring/
Summer 2025 show** during Paris Fashion Week.
```

RESULTS (correctness from the GPT-4o judge)

GPT-6 Luna, no tools	Miu Miu; 2024; the Miu Miu Spring/Summer 2025 show ✗
GPT-6 Luna + OpenVLHarness	**the Hermès show, Paris Fashion Week Fall/Winter 2025** ✓
Ground truth	StreetStyle Winter 2025 Hermes

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 4 · **Step 1** Image Search

WHAT HAPPENS Reverse image search on the photograph returns no relevant matches.

ARGUMENTS

```
{
  "image_id":  "input_image_1"
}
```

MODEL (text emitted with this call)

> *I'll use reverse-image search to identify the street-style photo or event source, which may reveal the represented brand and event details.*

TOOL OUTPUT

```
Tool `Image_Search_Tool` (tool_call_id=call_x1uncz6TEwkbHACyGoWvL0AL) returned:
{"ok":  true, "result":  "[Image Search | Visual Matches]
NO RELEVANT MATCHES

Why each raw match was rejected:
All 10 candidates show studio product photos of jackets, which visually mismatch the QUERY
image's street scene.  Their text metadata correctly describes the products but is
irrelevant to the QUERY image's context.  No candidate image matches the QUERY image's
subject or environment.

Suggested next actions:
- Image_Search_Tool(image_id=input_image_1)
- Text_Search_Tool(query='Paris street scene with woman in beige jacket', search_type=
images)", "error":  null}
```

Example 4 · **Step 2** Text Search

> **WHAT HAPPENS** The agent describes the outfit in words and runs an image search with that description. The top result is a street-style gallery from outside the Hermès show at Paris Fashion Week Fall/Winter 2025.

ARGUMENTS

```
{
  "query":  "Paris Fashion Week street style woman beige jacket orange scarf paper shopping
bag Paris black boots",
  "search_type":  "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_eGfMUoOe3PcNico2aru6bQVC) returned:
Image refs:  tool_generated_image_1, tool_generated_image_2, tool_generated_image_3
[Search Results | Type:  images]
1.  Paris Fashion Week FW 2025 - What they Wore to Hermes - FunkyForty
   Image:  [tool_generated_image_1]
   Source:  FunkyForty [webpage_url_1]
   Size:  1707x2560
2.  Showgoers Embraced the Colors of Fall for Day 5 of Paris Fashion Week - Fashionista
   Image:  [tool_generated_image_2]
   Source:  Fashionista [webpage_url_2]
   Size:  2560x3840
```

[1 more results not shown (3 in total).]





Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 3 new images in memory (tool_generated_image_1 to tool_generated_
image_3, from Text_Search_Tool); registers 3 new webpage URLs in memory (webpage_url_1 to
webpage_url_3, from Text_Search_Tool).
```

Example 4 · **Step 3** Zoom In

WHAT HAPPENS A zoom on the lettering behind the woman, which the agent takes to be her shopping bag.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [468, 477, 551, 772],
  "padding":  70
}
```

MODEL (text emitted with this call)

> *I'll inspect the lettering on the woman's shopping bag; it could provide a clue to the brand associated with her.*

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_7WEex9earQgiebwQJQxRu51z) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_4.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_4, from Zoom_In_Tool).
```

Example 4 · **Step 4** OCR

WHAT HAPPENS OCR reads the lettering, a municipal slogan on a Paris litter bag, not a brand.

ARGUMENTS

```
{
  "image_id":  "tool_generated_image_4"
}
```

TOOL OUTPUT

```
OCR_Tool transcribed the following text from the image:

ENSMLI
RENOIS
PARI
PROPIE
PARIS
```

Example 4 · **Step 5** Webpage Visit

WHAT HAPPENS A visit to the gallery page confirms the event: street style at the Hermès show, Paris Fashion Week Fall/Winter 2025.

ARGUMENTS


```
{
  "urls":  ["webpage_url_1"],
  "goal":  "Find whether the matching woman is shown and identify her outfit brand/
represented brand, the Paris Fashion Week year and event name/season."
}
```


MODEL (text emitted with this call)

> *The person was attending the Hermès show. The event was Paris Fashion Week Fall/Winter 2025.*

MODEL (text emitted with this call)

> *The person was attending the Hermès show. The event was Paris Fashion Week Fall/Winter 2025.*

TOOL OUTPUT


```
Tool `Webpage_Visit_Tool` (tool_call_id=call_ax7zd7wE1WPQ3f3IsEkgRq9v) returned:
{"ok":  true, "result":  "The useful information in https://funkyforty.com/
paris-fashion-week-fw-2025-what-they-wore-to-hermes/ for user goal Find whether the
matching woman is shown and identify her outfit brand/represented brand, the Paris Fashion
Week year and event name/season.  as follows:

Evidence in page:
Paris Fashion Week FW 2025 - What they Wore to Hermes.  The article discusses street style
at Hermes during Paris Fashion Week FW 2025, including styling ideas and bag trends, with
photos taken by Yvonne Aeberhard Stutz.  The event is clearly identified as Paris Fashion
Week Fall/Winter 2025.

Summary:
The webpage documents street style at Paris Fashion Week Fall/Winter 2025, specifically
focusing on what attendees wore to the Hermes show.  While it does not explicitly name a
specific woman, it references the event, season, and brand (Hermes) as requested in the
goal.

", "error":  null}
```


What to notice. When reverse image search fails, the agent turns the photograph into words and searches with them, which finds the exact street-style gallery the photo comes from. The zoom and OCR in the middle of the trajectory read an unrelated sign, and the agent is not misled by it: the answer rests on the gallery page.

Trajectory: `GPT6-Results/gpt-6-luna/apiagentwithtools/..._20260923_211245_default/MMSearchPlus/full_trajectories/161.json`

END OF EXAMPLE 4

Example 5 Dataset: MMSearch-Plus Model: Qwen3-VL-32B

Overview (our summary)

Task. The question, in Chinese, asks: how many years apart are the famous Great-Depression-era event commemorated in the first image and the event shown in the second? The first image is a rusted steel monument in a park; the second is a black-and-white photograph of a fight.
How the agent solves it. It identifies each image with reverse image search and dates each event from the web: the Ford Hunger March of 7 March 1932, and the Ford Strikers Riot of 3 April 1941. One page visit fails with a server error, and the agent recovers with a text search. It answers 9 years. The model reasons in Chinese throughout.

INPUT IMAGES



input_image_1



input_image_2

QUESTION (as sent to the model)

```
Question:  图1中发生在大萧条时代的著名历史事件，和图2中的事件相隔多少年?
```

RESULTS (no run of this model without tools is available for this question)

Qwen3-VL-32B + OpenVLHarness	**9**年 (9 years) ✓
Ground truth	9

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
- input_image_2:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 5 · **Step 1** Image Search

MODEL (text emitted with this call)

> *<thinking>*我需要先识别图*1*中的地点和相关的历史事件。从图*1*的墙上的文字*"FORT STREET BRIDGE PARK"*可以推断出这是位于美国密歇根州底特律市的一个公园。这个公园与底特律在工业时代的历史有关。然后，我需要确定图*2*中的历史事件是什么，并计算两个事件之间的时间间隔。
> 首先，我会使用图像搜索工具来确认图*1*中的雕塑或地标是否与某个特定的历史事件相关联。*</thinking>*

ARGUMENTS

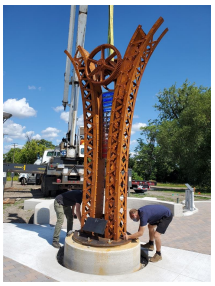
```
{
  "image_id": "input_image_1"
}
```


TOOL OUTPUT


```
Tool `Image_Search_Tool` (tool_call_id=chatcmpl-tool-89abdf6fe5e37463) returned:
Image refs: tool_generated_image_1, tool_generated_image_2, tool_generated_image_3
[Image Search | Visual Matches]
1. Fort Street Bridge Park chronicles labor and environmental history
   Source: planetdetroit.org/2024/03/fort-street-bridge-park [webpage_url_1]
   Image: [tool_generated_image_1]
2. MotorCities - Fort Street Bridge Interpretive Park
   Source: motorcities.org/fortstreet [webpage_url_2]
   Image: [tool_generated_image_2]
```


[1 more results not shown (3 in total).]






Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)


```
Memory update: registers 3 new images in memory (tool_generated_image_1 to tool_generated_image_3, from Image_Search_Tool); registers 3 new webpage URLs in memory (webpage_url_1 to webpage_url_3, from Image_Search_Tool).
```

Example 5 · **Step 2** Webpage Visit

MODEL (text emitted with this call)

> *<thinking>*根据图像搜索结果，图*1*显示的是位于底特律的*Fort Street Bridge Park*的雕塑。该公园纪念了底特律的劳工和环境历史。为了找到与大萧条时期相关的具体历史事件，我将访问其中一个来源网页以获取更多信息。*</thinking>*

ARGUMENTS


```
{
  "urls": ["webpage_url_1"],
  "goal": "Find out which historical event during the Great Depression is commemorated at Fort Street Bridge Park."
}
```


TOOL OUTPUT


```
Tool `Webpage_Visit_Tool` (tool_call_id=chatcmpl-tool-bbf896e936550bcb) returned:
{"ok": true, "result": "The useful information in https://planetdetroit.org/2024/03/fort-street-bridge-park/ for user goal Find out which historical event during the Great Depression is commemorated at Fort Street Bridge Park. as follows: \n\nEvidence in page: \nSigns explain why this sculpture and park are here: to honor the memory of protesters who met on this very spot on March 7, 1932, before marching up Miller Road to the massive Ford Rouge River Complex located in the adjacent city of Dearborn. ... This crowd of several thousand marched up the road on one of the coldest days of winter. They were greeted at the Dearborn border with clouds of tear gas, jets of cold water and a shower of bullets. It was then that the Ford Hunger March became the Ford Massacre.\n\nSummary: \nFort Street Bridge Park commemorates the Ford Hunger March of March 7, 1932, a pivotal labor protest during the Great Depression in which unemployed Ford workers marched to demand better working conditions and an end to discrimination, only to be violently suppressed by Ford security and Dearborn police | an event that became known as the Ford Massacre.\n\n", "error": null}
```

Example 5 · **Step 3** Image Search

MODEL (text emitted with this call)

> *<thinking>*现在我知道了图*1*中的事件是*1932*年的福特饥饿游行（*Ford Hunger March*），也被称为福特大屠杀（*Ford Massacre*）。接下来，我需要分析图*2*中的事件并确定其发生时间。图*2*是一张黑白照片，描绘了一群人正在打斗，其中一人举着棍子。这看起来像一个历史性的暴力事件。我将使用图像搜索工具来识别这张图片所代表的具体事件。*</thinking>*

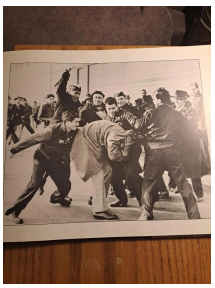
ARGUMENTS

```
{
  "image_id": "input_image_2"
}
```

TOOL OUTPUT

```
Tool `Image_Search_Tool` (tool_call_id=chatcmpl-tool-924377b36e8efdaf) returned:
Image refs: tool_generated_image_4, tool_generated_image_5, tool_generated_image_6
[Image Search | Visual Matches]
1. Ford Strikers Riot - A black and white photo of a group of men fighting - PICRYL -
Public Domain Media Search Engine Public Domain Search
   Source: picryl.com/media/ford-strikers-riot-a4a7fb [webpage_url_4]
   Image: [tool_generated_image_4]
2. Moments The Pulitzer Prize Photographs 1942-1977 Sheryle John Leekley Vtg Book -
very-good · Whatnot: Shop, Sell, Connect
   Source: whatnot.com/listing/TGlzdGluZ05vZGU6NzAzNzkzMTEy [webpage_url_5]
   Image: [tool_generated_image_5]
```

[1 more results not shown (3 in total).]


Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 3 new images in memory (tool_generated_image_4 to tool_generated_
image_6, from Image_Search_Tool); registers 3 new webpage URLs in memory (webpage_url_4 to
webpage_url_6, from Image_Search_Tool).
```

Example 5 · **Step 4** Webpage Visit

MODEL (text emitted with this call)

> *<thinking>*图像搜索结果显示，图*2*中的事件被标识为“福特罢工者骚乱”（*Ford Strikers Riot*）。为了准确确定这一事件的发生时间，我需要查阅更详细的信息。我将访问*PICRYL*网站上提供的链接，因为它是直接描述该事件的可靠来源。*</thinking>*

ARGUMENTS

```
{
  "urls": ["webpage_url_4"],
  "goal": "Determine the exact date of the Ford Strikers Riot shown in the image."
}
```

TOOL OUTPUT

```
Tool `Webpage_Visit_Tool` (tool_call_id=chatcmpl-tool-a3db75968c467ffc) returned:
{"ok": true, "result": "The useful information in https://picryl.com/media/
ford-strikers-riot-a4a7fb for user goal Determine the exact date of the Ford Strikers Riot
```

```
shown in the image.  as follows:  \n\nEvidence in page:  \nFailed to fetch https://
picryl.com/media/ford-strikers-riot-a4a7fb (HTTP 500, transient) after 3 attempts Detail:
500 Server Error:  Internal Server Error for url:  https://scrape.serper.dev/.  Suggestion:
source temporarily unavailable; try again later or pick a different URL.\n\nSummary:
\nFETCH FAILED.\n\nReason:  Failed to fetch https://picryl.com/media/
ford-strikers-riot-a4a7fb (HTTP 500, transient) after 3 attempts Detail:  500 Server Error:
Internal Server Error for url:  https://scrape.serper.dev/.  Suggestion:  source
temporarily unavailable; try again later or pick a different URL.\nDiagnosis:  Transient
failure (5xx / 429 / timeout) after 3 retries.  The source MAY be back online in a minute;
retrying this URL once more after a short pause is reasonable.  If it fails again, switch
to a different source.\n\nSuggested next actions for the agent:\n\n - Tool switches (the
agent may also re-call this same Webpage_Visit_Tool on a DIFFERENT URL | nothing is excluded
just because the current URL failed):\n - Webpage_Visit_Tool(urls=[<a different webpage_url_
N>], goal=...)\n Try a different URL from the same search results.  If the prior search
had multiple results and only th
```

[The remaining 541 characters of this output are not shown (2,041 in total).]

Example 5 · **Step 5** **Text Search**

MODEL (text emitted with this call)

> *<thinking>*由于无法访问*PICRYL*网站，我需要尝试另一个来源来确定福特罢工者骚乱的确切日期。我将使用文本搜索工具来查找关于“福特罢工者骚乱”的信息，特别是它的发生时间。 *</thinking>*

ARGUMENTS

```
{
  "query":  "Ford Strikers Riot date",
  "search_type":  "search"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=chatcmpl-tool-9b84597577bb0a7f) returned:
{"ok":  true, "result":  "[Search Results | Type:  search]\n1.  File:Ford Strikers Riot.jpg
- Wikimedia Commons\n Source:  commons.wikimedia.org/wiki/File:Ford_Strikers_Riot.jpg [tool_
generated_image_7]\n Date, Taken on 3 April 1941.  Source, The Detroit News, The legacy of
Henry ...  File:Ford Strikers Riot.png · Category:1941-04-03.  File usage on other
wikis.\n2.  Unemployed Detroit auto workers conduct Hunger March to protest ...\n Source:
nvdatabase.swarthmore.edu/content/unemployed-detroit-auto-workers-conduct-hun...  [webpage_
url_7]\n On 1 April 1941, Ford workers conducted a walkout in order to protest the firing
of eight union members.  Because of the strike, the River Rouge plant was ...\n3.  Ford
Strikers Riot - Wikipedia\n Source:  en.wikipedia.org/wiki/Ford_Strikers_Riot [webpage_url_
8]\n Ford Strikers Riot is a 1941 photograph which shows a strikebreaker getting beaten by
United Auto Workers (UAW) strikers.\n\n[Summary]\nThe Ford Strikers Riot occurred on April
3, 1941, as documented in a photograph taken by Milton Brooks and published by The Detroit
News, capturing a strikebreaker being beaten by United Auto Workers strikers.  A strike
began on April 1, 1941, after Ford fired eight union members, leading to a walkout and
escalating tensions.  The strike lasted about ten days, culminating in violence at the Ford
River Rouge Plant, with strikers using around 2,000 vehicles to block access to the plant.
The event is also referenced in Wikipedia and Wikimedia Commons as a pivotal moment in
American labor history, with the photograph later winning a Pulitzer Prize for
Photography.", "error":  null}
```

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_7, from Text_Search_
Tool); registers 2 new webpage URLs in memory (webpage_url_7 to webpage_url_8, from Text_
Search_Tool).
```

Example 5 · **Final answer**

MODEL

> *<thinking>*现在我已经知道：
> - 图*1*中的事件是**福特饥饿游行（*Ford Hunger March*）**，发生在 ***1932*年*3*月*7*日**。 - 图*2*中的事件是**福特罢工者骚乱（*Ford Strikers Riot*）**，发生在 ***1941*年*4*月*3*日**。
> 因此，这两个事件之间的年份差可以通过简单的减法计算得出：*1941 - 1932 = 9*年。
> 所以，图*1*和图*2*中的事件相隔*9*年。 *</thinking>*
> *<answer>9*年*</answer>*

What to notice. Each image is identified by its own reverse image search, and each event is dated from a web page, so the answer needs two independent chains of evidence and one subtraction. When a page fetch fails with a server error (Step 4), the harness reports the failure and suggests alternatives, and the model recovers with a text search instead of guessing the date.

Trajectory: `outputs/Qwen3-VL-32B-Instruct/baseagentwithtools/..._20260523_011655_default/MMSearchPlus/full_trajectories/234.json`

END OF EXAMPLE 5

Example 6 Dataset: RealX-Bench Model: GPT-6 Luna

Overview (our summary)

Task. A photograph shows a crowded skincare shelf. Which brand does the item at the far right of the top row belong to? The item is a tall white spray can whose label is far too small to read.
Why it is hard. Nothing legible identifies the product; without tools the model guesses AHC.
How the agent solves it. Zooms and OCR on the top shelf return only book spines and dates, and a reverse search of the whole shelf returns generic "shelfie" boards. The agent goes back to the image: it zooms on the bottles again, grounds them to see where each one sits, crops tightly, and reverse-searches the crop, which matches Miyuequan (Miyue Spring) hydrating spray.

INPUT IMAGES



input_image_1

QUESTION (as sent to the model)

```
Question:  Which brand does the item on the far right in the top row belong to?
```

WITHOUT TOOLS (GPT-6 Luna's full response to the same question)

```
The item on the far right in the top row appears to be from AHC.
```

RESULTS (correctness from the GPT-4o judge)

GPT-6 Luna, no tools	AHC ✗
GPT-6 Luna + OpenVLHarness	**MYONS** (弥泉) ✓
Ground truth	Miyue Spring

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 6 · **Step 1** Zoom In

WHAT HAPPENS A first zoom on the top-right corner of the shelf.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [812, 0, 872, 178],
  "padding":  70
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_aS30iH3kdrzpsXHTthJAJebA) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_1.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_1, from Zoom_In_Tool).
```

Example 6 · Step 2 OCR

WHAT HAPPENS OCR on the crop reads only a date, from something other than the spray can.

ARGUMENTS

```
{
  "image_id":  "tool_generated_image_1"
}
```

TOOL OUTPUT

```
OCR_Tool transcribed the following text from the image:

1982年10月1日
```

Example 6 · Step 3 Zoom In

WHAT HAPPENS A zoom further to the right, along the edge of the shelf.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [962, 18, 999, 176],
  "padding":  70
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_FP7fQd52QBGnF4r6XelMVkCY) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_2.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_2, from Zoom_In_Tool).
```

Example 6 · **Step 4** OCR

> **WHAT HAPPENS** OCR there reads the titles of books next to the shelf, not the product.

ARGUMENTS

```
{
  "image_id":  "tool_generated_image_2"
}
```

TOOL OUTPUT

```
OCR_Tool transcribed the following text from the image:

3 中公物
2 符合的直文例
9 作例
1 毛的趣人生
```

Example 6 · **Step 5** Image Search

> **WHAT HAPPENS** A reverse search of the whole photograph returns generic Pinterest boards of beauty shelves.

ARGUMENTS

```
{
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Tool `Image_Search_Tool` (tool_call_id=call_Whh5HfzIXfILrrmtcCajNDKj) returned:
Image refs:  tool_generated_image_3, tool_generated_image_4, tool_generated_image_5
[Image Search | Visual Matches]
1.  Discover 130 Beauty Shelf and shelf full of beauty products ideas | beauty products on
a shelf, skincare shelfie ideas, korean skincare products on shelf and more
   Source:  pinterest.com/honeymaesh/beauty-shelf [webpage_url_1]
   Image:  [tool_generated_image_3]
2.  I'm a Recent Dermstore Convert|Here Are the 21 Items I Would Buy in Bulk
   Source:  es.pinterest.com/pin/266064290476051096 [webpage_url_2]
   Image:  [tool_generated_image_4]
```

[1 more results not shown (3 in total).]




Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 3 new images in memory (tool_generated_image_3 to tool_generated_
image_5, from Image_Search_Tool); registers 3 new webpage URLs in memory (webpage_url_1 to
webpage_url_3, from Image_Search_Tool).
```

Example 6 · **Step 6** Zoom In

WHAT HAPPENS The agent zooms back onto the row of spray cans.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [814, 26, 870, 178],
  "padding":  100
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_NbfAqx1ZRkUL0NVPM4WWiKL1) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_6.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_6, from Zoom_In_Tool).
```

Example 6 · **Step 7** Visual Grounding

WHAT HAPPENS Grounding “shampoo bottle” on that crop separates the four tall bottles, so the agent can see which one is at the far right.

ARGUMENTS

```
{
  "query": "shampoo bottle",
  "image_id": "tool_generated_image_6"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 4 objects for the prompt 'shampoo bottle'. Their
corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: ```json
[
    {"bbox_2d": [0, 144, 18, 615], "label": "shampoo bottle"},
    {"bbox_2d": [16, 50, 197, 621], "label": "shampoo bottle"},
    {"bbox_2d": [198, 52, 376, 623], "label": "shampoo bottle"},
    {"bbox_2d": [376, 54, 554, 624], "label": "shampoo bottle"}
]```. The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_7.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_7, from Visual_
Grounding_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- segment_shampoo_bottle: dict (keys: pred_boxes, pred_scores, pred_masks, num_masks, orig_
img_h, orig_img_w)
```

Example 6 · **Step 8** Zoom In

WHAT HAPPENS A tighter zoom on the rightmost bottles.

ARGUMENTS

```
{
  "image_id": "input_image_1",
  "coords": [804, 37, 872, 176],
  "padding": 120
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_azos4cSBGPDQ4vfi9c2sTq93) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_8.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_8, from Zoom_In_Tool).
```

Example 6 · **Step 9** Zoom In

> **WHAT HAPPENS** A final tight crop around the target can.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [805, 25, 875, 175],
  "padding":  100
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_Nq6FPXZJ2SVXf38UvzL0sBlU) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_9.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_9, from Zoom_In_Tool).
```

Example 6 · **Step 10** OCR

> **WHAT HAPPENS** OCR on the tight crop again reads only a date.

ARGUMENTS

```
{
  "image_id":  "tool_generated_image_9"
}
```


TOOL OUTPUT


```
OCR_Tool transcribed the following text from the image:

2015年1月1日
```


Example 6 · **Step 11** Image Search

WHAT HAPPENS Reverse image search on the tight crop. Its second match is Miyuequan Hydrating Spray, the product on the shelf.

ARGUMENTS


```
{
  "image_id":  "tool_generated_image_9"
}
```


MODEL (text emitted with this call)

> *MYONS (弥泉)*

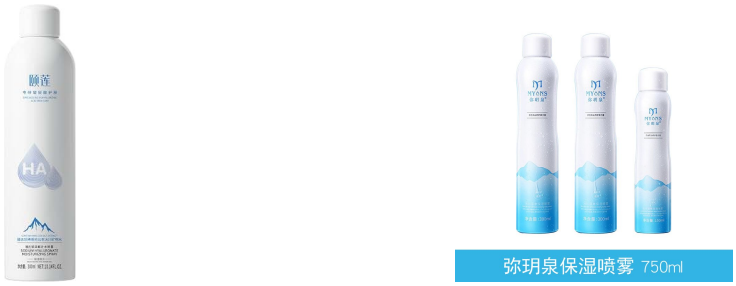
MODEL (text emitted with this call)

> *MYONS (弥泉)*

TOOL OUTPUT


```
Tool `Image_Search_Tool` (tool_call_id=call_zqaG64IKUZe4rIVRkEfufdxn) returned:
Image refs:  tool_generated_image_10, tool_generated_image_11, tool_generated_image_12
[Image Search | Visual Matches]
1.  Amazon.com:  YILIAN Face Mist Natural Facial Spray with HA, Soothing and Calming
Sensitive Skin, 10.14 fl.  oz.
   Source:  amazon.com/clp/B0DTT9SV1Y [webpage_url_4]
   Image:  [tool_generated_image_10]
2.  Fast delivery][Hot selling in China] Miyuequan Hydrating Spray Moisturizing for Women
Soothing Skin Pre-Makeup Glacier Hot Spring Small Molecule Toner Miyuequan | Daraz.com.np
   Source:  daraz.com.np/products/
fast-deliveryhot-selling-in-china-miyuequan-hydrating-s...  [webpage_url_5]
   Image:  [tool_generated_image_11]
```


[1 more results not shown (3 in total).]



Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)


```
Memory update:  registers 3 new images in memory (tool_generated_image_10 to tool_generated_
image_12, from Image_Search_Tool); registers 3 new webpage URLs in memory (webpage_url_4 to
webpage_url_6, from Image_Search_Tool).
```


What to notice. When the whole-image search fails, the agent does not keep searching blindly: it returns to the image, uses grounding to isolate the target among similar bottles, and turns a tight crop into the search query. Perception and search alternate several times, and the product is identified only once the crop is tight enough for the reverse search to match it.

Trajectory: `GPT6-Results/gpt-6-luna/apiagentwithtools/..._20260924_014712_default/RealXBench/full_trajectories/140.json`

END OF EXAMPLE 6

Example 7 Dataset: RealX-Bench Model: GPT-6 Sol

Overview (our summary)

Task. A makeup organizer holds brushes and cosmetics. Which brand is the liquid foundation at the far right? Its label is unreadable; its most distinctive feature is a crystal-faceted cap.
Why it is hard. Without tools the model guesses Dior.
How the agent solves it. OCR reads nothing, and reverse-searching a crop of the bottle finds no match. The agent then searches by description ("faceted crystal cap"), tries a Jill Stuart hypothesis and drops it, and reverse-searches a tighter crop. That search's summary names the bottle as KATOKATO, and a targeted image search returns a KATO-KATO foundation whose cap matches the crop.
Caveat. The brand name first appears in the reverse-search tool's own summary of the crop, so the decisive step is partly inside the tool.

INPUT IMAGES



input_image_1

QUESTION (as sent to the model)

```
Question:  Which brand does the liquid foundation on the far right on the storage shelf in
the picture belong to?
```

WITHOUT TOOLS (GPT-6 Sol's full response to the same question)

```
The liquid foundation on the far right appears to be **Dior**.
```

RESULTS (correctness from the GPT-4o judge)

GPT-6 Sol, no tools	Dior ✗
GPT-6 Sol + OpenVLHarness	**KATO-KATO** ✓
Ground truth	KATO

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 7 · **Step 1** OCR

WHAT HAPPENS OCR on the whole photograph finds no readable text.

ARGUMENTS

```
{
  "image_id":  "input_image_1"
}
```

MODEL (text emitted with this call)

> *I'll check the label on the rightmost foundation bottle on the shelf.*

TOOL OUTPUT

```
OCR_Tool did not transcribe any readable text.
```

Example 7 · **Step 2** Zoom In

WHAT HAPPENS A zoom on the rightmost bottle.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [423, 495, 485, 585],
  "padding":  70
}
```

MODEL (text emitted with this call)

> *The label wasn't readable at full size, so I'll inspect the rightmost bottle more closely.*

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_zZ2NiO8lBIDogDGCwUzWI6ZD) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_1.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_1, from Zoom_In_Tool).
```

Example 7 · **Step 3** Image Search

WHAT HAPPENS Reverse image search on that crop finds no relevant match.

ARGUMENTS

```
{
  "image_id":  "tool_generated_image_1"
}
```


MODEL (text emitted with this call)

> *I'll compare the bottle's distinctive cap and shape with product images to confirm the brand.*

TOOL OUTPUT


```
Tool `Image_Search_Tool` (tool_call_id=call_OoscIm9kAfd999QA3dwuSAnZ) returned:
{"ok":  true, "result":  "[Image Search | Visual Matches]
NO RELEVANT MATCHES

Why each raw match was rejected:
All candidates show single cosmetic products (nail polish or foundation) in isolation,
while the QUERY image shows a multi-tiered makeup organizer with brushes and cosmetics.
The image-text alignment is correct for each candidate but irrelevant to the QUERY image's
subject.  The text metadata is domain-relevant (beauty products) but fails to match the
QUERY image's specific visual subject.

Suggested next actions:
- Image_Search_Tool(image_id=input_image_1)", "error":  null}
```

Example 7 · **Step 4** Zoom In

> **WHAT HAPPENS** A tighter zoom on the bottle's label and cap.

ARGUMENTS


```
{
  "image_id":  "input_image_1",
  "coords":  [432, 511, 474, 570],
  "padding":  50
}
```


MODEL (text emitted with this call)

> *I'll look closer at the bottle's lettering and cap.*

TOOL OUTPUT


```
Tool `Zoom_In_Tool` (tool_call_id=call_IaM1kjcmpzK9Jytg2996KDA3) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_2.
```




Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)


```
Memory update:  registers 1 new image in memory (tool_generated_image_2, from Zoom_In_Tool).
```

Example 7 · **Step 5** OCR

> **WHAT HAPPENS** OCR on the tight crop still reads nothing.

ARGUMENTS

```
{
  "image_id":  "tool_generated_image_2"
}
```

TOOL OUTPUT

```
OCR_Tool did not transcribe any readable text.
```

Example 7 · **Step 6** Text Search

WHAT HAPPENS A text search that describes the bottle (faceted silver cap, beige liquid) returns lists of luxury foundations.

ARGUMENTS

```
{
  "query":  "foundation bottle ornate faceted silver cap rounded glass beige foundation
black base luxury brand",
  "search_type":  "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_BGJ52afbD0ZHnyo2fgnt4UGG) returned:
Image refs:  tool_generated_image_3, tool_generated_image_4, tool_generated_image_5
[Search Results | Type:  images]
1.  Top 7 Luxury Foundations:  Flawless, Long-Lasting & Easy to Apply
   Image:  [tool_generated_image_3]
   Source:  Lemon8 [webpage_url_1]
   Size:  1440x1920
2.  Top 7 Luxury Foundations:  Flawless, Long-Lasting & Easy to Apply
   Image:  [tool_generated_image_4]
   Source:  Lemon8 [webpage_url_1]
   Size:  1440x1920
```

[1 more results not shown (3 in total).]



Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 3 new images in memory (tool_generated_image_3 to tool_generated_
image_5, from Text_Search_Tool); registers 1 new webpage URL in memory (webpage_url_1, from
Text_Search_Tool).
```

Example 7 · **Step 7** Text Search

WHAT HAPPENS A second descriptive search returns nothing relevant.

ARGUMENTS

```
{
  "query":  "foundation bottle clear faceted jewel crystal cap silver beige liquid round
short bottle black base",
  "search_type":  "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_S6nR1Mt5MOIn4aui2uN06iyd) returned:
{"ok": true, "result": "[Search Results | Type: images]
NO RELEVANT RESULTS

Why each raw result was rejected:
All candidates show foundation bottles that are either square, cylindrical, or lack the
specified faceted crystal design, silver cap, or black base. Image 4 is the closest but
lacks the cap color and base detail. Image 9 has beige liquid but a green cap and is not
clear or faceted. Image 5 and 8 show frosted bottles, not clear or faceted. Image 1, 2,
3, 6, 7 show square bottles with black caps, not round or faceted. Image 10 shows a tube,
not a bottle. All candidates fail on at least one key attribute (shape, cap color, base
color, faceted crystal design).

Suggested query rewrites:
- clear faceted glass foundation bottle with black cap and beige liquid
- jewel crystal foundation bottle round short black base
- refillable clear glass foundation bottle with silver cap

Suggested next actions:
- Text_Search_Tool(query="clear faceted glass foundation bottle with black cap and beige
liquid", search_type=images)
- Text_Search_Tool(query="jewel crystal foundation bottle round short black base", search_
type=images)
- Text_Search_Tool(query="refillable clear glass foundation bottle with silver cap", search_
type=images)", "error": null}
```

Example 7 · **Step 8** Text Search

WHAT HAPPENS The agent tests a hypothesis, Jill Stuart, whose products have crystal-style caps; the results are concealers, not this bottle.

ARGUMENTS

```
{
  "query": "Jill Stuart foundation bottle crystal ornate silver cap round beige",
  "search_type": "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_dE4zv498aQD0an48qVLBJ0jv) returned:
Image refs: tool_generated_image_6, tool_generated_image_7, tool_generated_image_8
[Search Results | Type: images]
1. Diamond Tips Concealer | PRODUCTS | JILL STUART Beauty Official Site
   Image: [tool_generated_image_6]
   Source: www.jillstuart-beauty.com [webpage_url_2]
   Size: 640x640
2. JILL STUART Base Makeup New Items | JILL STUART Beauty Official Site
   Image: [tool_generated_image_7]
   Source: www.jillstuart-beauty.com [webpage_url_3]
   Size: 750x1090
```

[1 more results not shown (3 in total).]





Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 3 new images in memory (tool_generated_image_6 to tool_generated_
image_8, from Text_Search_Tool); registers 3 new webpage URLs in memory (webpage_url_2 to
webpage_url_4, from Text_Search_Tool).
```

Example 7 · **Step 9** Text Search

WHAT HAPPENS A second Jill Stuart search does not match the bottle either, so the agent drops the hypothesis.

ARGUMENTS

```
{
  "query":  "Jill Stuart liquid foundation glass bottle ornate crystal cap black base
foundation",
  "search_type":  "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_IjxBHLUO1w76MLNjNAudQW3e) returned:
Image refs:  tool_generated_image_9, tool_generated_image_10, tool_generated_image_11
[Search Results | Type:  images]
1.  JILL STUART Diamond Tips Concealer Cover type 5.3mL Japan
   Image:  [tool_generated_image_9]
   Source:  eBay [webpage_url_5]
   Size:  923x1200
2.  JILL STUART Base Makeup New Items & Limited Items | NEW ITEM | JILL STUART Beauty
Official Site
   Image:  [tool_generated_image_10]
   Source:  JILL STUART [webpage_url_6]
   Size:  650x618
```

[1 more results not shown (3 in total).]





Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 3 new images in memory (tool_generated_image_9 to tool_generated_
image_11, from Text_Search_Tool); registers 3 new webpage URLs in memory (webpage_url_5 to
webpage_url_7, from Text_Search_Tool).
```

Example 7 · **Step 10** Text Search

WHAT HAPPENS Another descriptive search returns only empty stock bottles.

ARGUMENTS

```
{
  "query":  "foundation bottle \"black\" \"crystal\" cap \"silver\" round liquid
foundation",
  "search_type":  "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_xM5R3uof4FvmZ1yv5tkzAkW6) returned:
Image refs:  tool_generated_image_12, tool_generated_image_13, tool_generated_image_14
[Search Results | Type:  images]
1.  30ml Custom Logo Oem Plastic Empty Cosmetic Liquid Foundation Bottle Round Silver Clear
Foundation Stick With Huge Brush Wand - Buy Product on Alibaba.com
   Image:  [tool_generated_image_12]
   Source:  Alibaba.com [webpage_url_8]
   Size:  2268x2268
```

```
2.  30ml Custom Logo Oem Plastic Empty Cosmetic Liquid Foundation Bottle Round Silver Clear
Foundation Stick With Huge Brush Wand - Buy Product on Alibaba.com
   Image:  [tool_generated_image_13]
   Source:  Alibaba.com [webpage_url_8]
   Size:  800x800
```

[1 more results not shown (3 in total).]





Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 3 new images in memory (tool_generated_image_12 to tool_generated_
image_14, from Text_Search_Tool); registers 2 new webpage URLs in memory (webpage_url_8 to
webpage_url_9, from Text_Search_Tool).
```

Example 7 · **Step 11** Image Search

WHAT HAPPENS A reverse search on the tight crop from Step 4. The tool's summary identifies a ZOYA nail polish and "a KATOKATO foundation bottle".

ARGUMENTS

```
{
  "image_id":  "tool_generated_image_2"
}
```

TOOL OUTPUT

```
Tool `Image_Search_Tool` (tool_call_id=call_iL45p6mk8iTJWkV0ud1xLoi9) returned:
Image refs:  tool_generated_image_15, tool_generated_image_16, tool_generated_image_17
[Image Search | Visual Matches]
1.  Zoya Bridal Bliss Collection (Part II) Swatch and Review - LacquerExpression
   Source:  lacquerexpression.com/2018/06/mainstream-swatch-and-review-zoya_28.html
[webpage_url_10]
   Image:  [tool_generated_image_15]
2.  Amazon.com:  ZOYA Nail Polish, Maxine, 0.5 fl.  oz.  :  Beauty & Personal Care
   Source:  amazon.com/ZOYA-Nail-Polish-Maxine-0-5/dp/B07HHN2N6C [webpage_url_11]
   Image:  [tool_generated_image_16]
```

[1 more results not shown (3 in total).]





Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 3 new images in memory (tool_generated_image_15 to tool_generated_
image_17, from Image_Search_Tool); registers 3 new webpage URLs in memory (webpage_url_10 to
webpage_url_12, from Image_Search_Tool).
```

Example 7 · **Step 12** Text Search

WHAT HAPPENS A targeted image search for KATOKATO foundation returns a KATO-KATO product whose crystal cap matches the crop.

ARGUMENTS

```
{
  "query":  "\"KATOKATO\" liquid foundation crystal cap bottle",
  "search_type":  "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_ILCAxaED23QcWvUq1eDpGdmQ) returned:
Image refs:  tool_generated_image_18
[Search Results | Type:  images]
1.  KATO-KATO { Chic Decent
   Image:  [tool_generated_image_18]
   Source:  Chic Decent [webpage_url_13]
   Size:  645x645

[Summary]
The primary candidate (rank 1) from Chic Decent clearly shows the KATO-KATO liquid
foundation with its distinctive crystal cap, matching the query exactly.  The second
candidate (rank 2) displays a KATO-KATO product but it is a 'Misty Project' spray with a
black cap, not a liquid foundation.  The third candidate (rank 3) is a 'Cloudy to Misty
Lightweight Setting Spray' also with a black cap, confirming it is not the requested
foundation.  All three sources are from Chic Decent or The Hype Cosmetics Outlet, both
retailing KATO-KATO products, but only the first image correctly depicts the liquid
foundation with the crystal cap as specified in the query.
tool_generated_image_18
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_18, from Text_Search_
Tool); registers 1 new webpage URL in memory (webpage_url_13, from Text_Search_Tool).
```

Example 7 · Final answer

WHAT HAPPENS The agent answers KATO-KATO, which matches the ground truth, KATO.

MODEL

```
<answer>KATO-KATO</answer>
```

What to notice. With no readable text, the agent searches by what it can see (the faceted cap), states and rejects a brand hypothesis, and returns to reverse image search on its own crop. The final targeted search lets it check the candidate brand against the cap it saw, rather than accept the name on trust.

Trajectory: `GPT6-Results/gpt-6-sol/apiagentwithtools/..._20260923_191423_default/RealXBench/full_trajectories/2.json`

END OF EXAMPLE 7

Example 8 Dataset: FVQA Model: Qwen3-VL-32B

Overview (our summary)

Task. A photograph shows two hikers resting with their backpacks. What awards has this product won?
Why it is hard. The product is not named anywhere, and the logos on the backpacks are too small to read.
How the agent solves it. It grounds the two backpacks, tries to read each with zoom and OCR, and, when that fails, reverse-searches the crop of the green backpack, which identifies it as a Deuter Speed Lite Pro. A product page mentions no award, so it searches for awards and verifies the result on the manufacturer's press release: the ISPO Award 2024. Nine calls use six different tools.

INPUT IMAGES



input_image_1

QUESTION (as sent to the model)

```
Question:  What awards has this product won?
```

RESULTS (no run of this model without tools is available for this question)

Qwen3-VL-32B + OpenVLHarness	**The Deuter Speed Lite Pro backpack has won the ISPO Award 2024** ✓
Ground truth	ISPO Award 2024

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 8 · **Step 1** Visual Grounding

MODEL (text emitted with this call)

> *<thinking>The image shows two people sitting outdoors with backpacks, likely hiking or camping. There is no clear indication of a specific product being advertised or labeled in the image. Without identifying the exact product, it's impossible to determine any awards it may have won. I need to identify the product first.</thinking>*

ARGUMENTS

```
{
  "query":  "backpack",
  "image_id":  "input_image_1"
```

```
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 2 objects for the prompt 'backpack'.  Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  {'object 1':  [199.49, 423.01,
407.82, 875.1], 'object 2':  [552.17, 550.46, 684.45, 951.58]}.  The object numbers
correspond to the labeled masks rendered in the following image, which you can refer to as
tool_generated_image_1.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_1, from Visual_
Grounding_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- segment_backpack:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 8 · **Step 2** Zoom In

MODEL (text emitted with this call)

> *<thinking>I have identified two backpacks in the image. To determine which product might have won awards, I need to examine them more closely. I'll start by zooming in on the purple backpack (labeled as object 1) to see if there are any visible brand names, logos, or labels that could help identify the product.</thinking>*

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [199, 423, 408, 875],
  "padding":  50
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=chatcmpl-tool-92335cf0b5261d44) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_2.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_2, from Zoom_In_Tool).
```

Example 8 · Step 3 OCR

MODEL (text emitted with this call)

> *<thinking>I've zoomed in on the purple backpack, but the details are still not clear enough to read any text or logos. I'll now attempt to use OCR to extract any text from this zoomed-in image to identify the brand or model of the backpack.</thinking>*

ARGUMENTS

```
{
  "image_id":  "tool_generated_image_2"
}
```

TOOL OUTPUT

```
OCR_Tool did not transcribe any readable text.
```

Example 8 · Step 4 Zoom In

MODEL (text emitted with this call)

> *<thinking>The OCR tool failed to extract any text from the zoomed-in image of the purple backpack. I will now zoom in on the green backpack (labeled as object 2) to check for any visible branding or labels.</thinking>*

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [552, 550, 684, 951],
  "padding":  50
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=chatcmpl-tool-bd7bc329779eaddf) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_3.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_3, from Zoom_In_Tool).
```

Example 8 · **Step 5** OCR

MODEL (text emitted with this call)

> *<thinking>I've zoomed in on the green backpack, and it appears to have some text or a logo on it. I will now use OCR to extract any readable text from this zoomed-in image to identify the brand or model.</thinking>*

ARGUMENTS

```
{
  "image_id":  "tool_generated_image_3"
}
```

TOOL OUTPUT

```
OCR_Tool transcribed the following text from the image:

_
```

Example 8 · **Step 6** Image Search

MODEL (text emitted with this call)

> *<thinking>Despite zooming in and attempting OCR, no readable text was extracted from either backpack. Since the visual inspection hasn't yielded any identifiable brand or product name, I will now use an image search to identify the backpacks based on their appearance.</thinking>*

ARGUMENTS

```
{
  "image_id":  "tool_generated_image_3"
}
```

TOOL OUTPUT

```
Tool `Image_Search_Tool` (tool_call_id=chatcmpl-tool-bd7ce98f66d6d010) returned:
Image refs:  tool_generated_image_4, tool_generated_image_5, tool_generated_image_6
[Image Search | Visual Matches]
1.  DEUTER Speed Lite Pro 25 Hiking Backpack in Mineral Grove | eBay Australia
   Source:  ebay.com.au/itm/388231206979 [webpage_url_1]
   Image:  [tool_generated_image_4]
2.  deuter Men's Speed Lite Pro 19 Hiking Backpack :  Amazon.ca:  Sports & Outdoors
   Source:  amazon.ca/deuter-Mens-Speed-Hiking-Backpack/dp/B0F79T5CY9 [webpage_url_2]
   Image:  [tool_generated_image_5]
```

[1 more results not shown (3 in total).]



Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 3 new images in memory (tool_generated_image_4 to tool_generated_
image_6, from Image_Search_Tool); registers 3 new webpage URLs in memory (webpage_url_1 to
webpage_url_3, from Image_Search_Tool).
```

Example 8 · **Step 7** Webpage Visit

MODEL (text emitted with this call)

> *<thinking>The image search results strongly indicate that the green backpack is a Deuter Speed Lite Pro model. Now, I will visit the Amazon.ca product page for the Deuter Speed Lite Pro 19 to gather information about any awards it may have won.</thinking>*

ARGUMENTS

```
{
  "urls": ["webpage_url_2"],
  "goal": "Find information about awards won by the Deuter Speed Lite Pro 19 hiking
backpack."
}
```

TOOL OUTPUT

```
Tool `Webpage_Visit_Tool` (tool_call_id=chatcmpl-tool-8cd49fe4cf0b84ab) returned:
{"ok": true, "result": "The useful information in https://www.amazon.ca/
deuter-Mens-Speed-Hiking-Backpack/dp/B0F79T5CY9 for user goal Find information about awards
won by the Deuter Speed Lite Pro 19 hiking backpack. as follows: \n\nEvidence in page:
\ndeuter Men's Speed Lite Pro 19 Hiking Backpack : Amazon.ca: Sports & Outdoors\n\nBrand:
deuter\n\n4.7 out of 5 stars (18)\n\n\ndeuter Men's Speed Lite Pro 19 Hiking Backpack\n\n\n
1. \n\n 2. \n 3. \n 4. \n 5. \n 6. \n 7. \n 8. \n\nLoading Images...\n\nDouble-tap
to zoom\n\nBrief content visible, double tap to read full content.\n\nFull content visible,
double tap to read brief content.\n\nSize: 19 L\n\nColour Name: Mineral-grove\n\n{\"
mobile_buybox_group_1\":[{\"displayPrice\":\"$243.61\",\"priceAmount\":243.61,\"
currencySymbol\":\"$\",\"integerValue\":\"243\",\"decimalSeparator\":\".\",\"
fractionalValue\":\"61\",\"symbolPosition\":\"left\",\"hasSpace\":false,\"
showFractionalPartIfEmpty\":true,\"offerListingId\":\"
6sWeWXGQzh2cpPC17FYwenyQPoo7llN1pV7pe4%2BwTvQ5FpoT51MwX7j4SSrR%2Fs2SWOjBZNZ3PkP0bR5B%2Bugzx%2FySTXozbx65fDQnTINYrDHi1pu
,\"locale\":\"en-CA\",\"buyingOptionType\":\"NEW\",\"aapiBuyingOptionIndex\":0}]}\n\nBrief
content visible, double tap to read full content.\n\nFull content visible, double tap to
read brief content.\n\n\nPurchase options and add-o
```

[The remaining 2,268 characters of this output are not shown (3,768 in total).]

Example 8 · **Step 8** Text Search

MODEL (text emitted with this call)

> *<thinking>The Amazon.ca product page for the Deuter Speed Lite Pro 19 hiking backpack does not mention any awards won by the product. I will now search for information about awards won by the Deuter Speed Lite Pro series or Deuter backpacks in general.</thinking>*

ARGUMENTS

```
{
  "query": "awards won by Deuter Speed Lite Pro backpack",
  "search_type": "search"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=chatcmpl-tool-a3e251071bbd98bb) returned:
{"ok": true, "result": "[Search Results | Type: search]\n1. The New deuter Speed Lite
Pro Wins ISPO Award for Lightweight ...\n Source: outdoorindustry.org/press-release/
the-new-deuter-speed-lite-pro-wins-ispo-awa... [webpage_url_4]\n The New deuter Speed Lite
Pro Wins ISPO Award for Lightweight Speed on the Trail · Unmatched Comfort and Stability ·
Quick Access on the Move ...\n2. New Deuter Speed Lite Pro 2025 Backpacks - Gear Tips Club
Blog\n Source: geartips.club/blog/en/new-deuter-speed-lite-pro-2025-backpacks [webpage_url_
5]\n ... earned the ISPO 2024 Award for the Speed Lite Pro. The new Deuter Speed Lite Pro
is even lighter. The new Deuter Speed Lite Pro surpasses ...\n3. Speed Lite Pro 28 SL |
black | M | 4046051170237 - deuter\n Source: deuter.com/us-en/speed-lite-pro-28-sl/
4046051170237 [webpage_url_6]\n This featherweight backpack weighs in at just 680 grams
(Speed Lite Pro 25) und 740 grams (Speed Lite Pro 28 SL). Which means you can put more
energy into ...\n\n[Summary]\nThe Deuter Speed Lite Pro backpack won the ISPO Award in 2024
for Lightweight Speed on the Trail, as confirmed by multiple sources including
outdoorindustry.org and geartips.club. The backpack is noted for being lighter than
previous models, with the Speed Lite Pro 25 weighing 680 grams and the 28 SL at 740 grams.
It features sustainable materials, improved breathability, and enhanced comfort for
fast-paced mountain adventures, with quick-access pockets and vest-style shoulder straps.
Additional reviews from CleverHiker and Gearjunkie highlight its pocket-rich design and
```

performance on trails, while Reddit users discuss its utility for day hikes and overnight treks.", "error": null}

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

Memory update: registers 3 new webpage URLs in memory (webpage_url_4 to webpage_url_6, from Text_Search_Tool).

Example 8 · **Step 9** Webpage Visit

MODEL (text emitted with this call)

> *<thinking>The search results confirm that the Deuter Speed Lite Pro backpack won the ISPO Award in 2024 for Lightweight Speed on the Trail. To verify this information, I will visit the outdoorindustry.org press release page.</thinking>*

ARGUMENTS

{
"urls": ["webpage_url_4"],
"goal": "Verify the award won by the Deuter Speed Lite Pro backpack as mentioned in the press release."
}

TOOL OUTPUT

Tool `Webpage_Visit_Tool` (tool_call_id=chatcmpl-tool-ba493f77370099c0) returned:
{"ok": true, "result": "The useful information in https://outdoorindustry.org/press-release/
the-new-deuter-speed-lite-pro-wins-ispo-award-for-lightweight-speed-on-the-trail/ for user goal Verify the award won by the Deuter Speed Lite Pro backpack as mentioned in the press release. as follows: \n\nEvidence in page: \nThe New deuter Speed Lite Pro Wins ISPO Award for Lightweight Speed on the Trail... deuter, a global leader in backpacks and outdoor equipment, celebrates the ISPO Award 2024 win for its new Speed Lite Pro backpack series. ... 'We're honored to receive the ISPO award for our new Speed Lite Pro series.' ... 'The new Speed Lite Pro impresses with its great fit, good ventilation and low weight, ' says the IPSO Award Jury.\n\nSummary: \nThe Deuter Speed Lite Pro backpack won the ISPO Award 2024 for Lightweight Speed on the Trail, as confirmed by deuter's press release and the award jury's statement, highlighting its lightweight design, comfort, and performance for speed hiking.\n\n", "error": null}

Example 8 · **Final answer**

MODEL

> *<thinking>The verified information confirms that the Deuter Speed Lite Pro backpack won the ISPO Award 2024 for Lightweight Speed on the Trail. This is the only award mentioned in the reliable sources consulted. Therefore, I can confidently state the award won by the product shown in the image.</thinking>*
> *<answer>The Deuter Speed Lite Pro backpack has won the ISPO Award 2024 for Lightweight Speed on the Trail.</answer>*

What to notice. The answer depends on first naming a product that no text in the image names. Perception tools narrow the photo down to the right object (grounding, zoom), OCR fails, and the model switches to reverse image search on its own crop, which identifies the product. It then does not stop at the first page that lacks the answer: it searches for the award directly and verifies it at the source.

Trajectory: `outputs/Qwen3-VL-32B-Instruct/baseagentwithtools/..._20260523_093605_default/FVQA_test/full_trajectories/fvqa_test_1649.json`

END OF EXAMPLE 8

Example 9 Dataset: BrowseComp-VL Model: Qwen3-VL-32B

Overview (our summary)

Task. A photograph shows the leaves and catkins of a flowering tree. From a riddle-like description (a common name that resembles an animal feature, a family that also contains a common Northern-Hemisphere genus, endemic to several provinces of southwestern China), name the regions of China where it is native.
How the agent approaches it. A reverse image search fails, so the agent hypothesizes a species, *Castanopsis argentea*, tests it with a text search and a Wikipedia visit, finds that the species is native to Java and Sumatra rather than China, rejects it, and searches again at the level of the genus.
Caveat. The species is never identified. The four provinces in the final answer first appear in the model's own reasoning, before any search supports them, and the searches only confirm that the genus is diverse in Yunnan. The answer matches the ground truth, but it is not fully grounded in the retrieved evidence.

INPUT IMAGES



input_image_1

QUESTION (as sent to the model)

```
Question:  In the image, this flowering plant species, often referred to by its common name
which resembles a distinctive animal feature, belongs to a family that also includes a
genus commonly found in temperate regions of the Northern Hemisphere.  This plant is
endemic to several provinces in southwestern China, where it thrives in a variety of
habitats.  Which regions of China is this plant native to?
```

RESULTS (no run of this model without tools is available for this question)

Qwen3-VL-32B + OpenVLHarness	**Yunnan, Guizhou, Sichuan, and Guangxi provinces in southwestern China** ✓
Ground truth	Sichuan, eastern Yunnan, northern Guangxi, and Guizhou

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 9 · **Step 1** Image Search

MODEL (text emitted with this call)

> *<thinking>The image shows a plant with green leaves and catkin-like structures, which resemble the common name "hazel" or "beech," possibly referring to a plant like *Corylus* (hazelnut) or another member of the Betulaceae family. However, the question specifically mentions a plant endemic to southwestern China, with a common name resembling a distinctive animal feature—this strongly suggests *Castanopsis*, commonly called "chestnut" or sometimes "horse chestnut" (though horse chestnut is typically *Aesculus*). But more likely, given the habitat and region, it could refer to *Castanopsis argentea* or similar species, which are part of the Fagaceae family (which also includes oaks and beeches).*
> *To identify the exact species and confirm its native regions in China, I need to perform a reverse image search to determine the precise plant species.</thinking>*

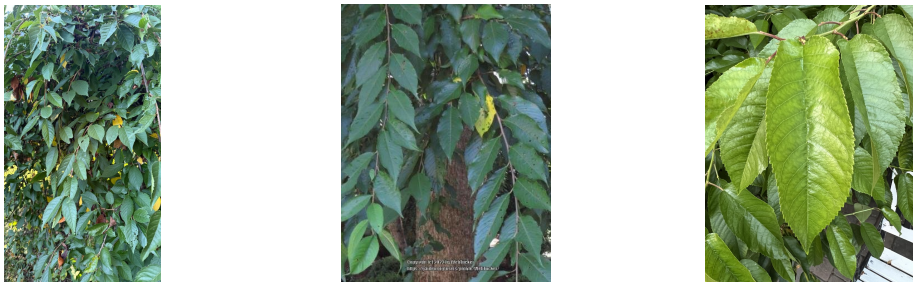
ARGUMENTS

```
{
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Tool `Image_Search_Tool` (tool_call_id=chatcmpl-tool-a90c981a4fce8386) returned:
Image refs:  tool_generated_image_1, tool_generated_image_2, tool_generated_image_3
[Image Search | Visual Matches]
1.  Wondering what this plant is in my new yard?  :  r/whatsthisplant
   Source:  reddit.com/.../wondering_what_this_plant_is_in_my_new_yard [webpage_url_1]
   Image:  [tool_generated_image_1]
2.  Is this Weeping Elm?  in the Plant ID forum - Garden.org
   Source:  garden.org/thread/view/164184/Is-this-Weeping-Elm [webpage_url_2]
   Image:  [tool_generated_image_2]
```

[1 more results not shown (3 in total).]



Images returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 3 new images in memory (tool_generated_image_1 to tool_generated_
image_3, from Image_Search_Tool); registers 3 new webpage URLs in memory (webpage_url_1 to
webpage_url_3, from Image_Search_Tool).
```

Example 9 · **Step 2** Text Search

MODEL (text emitted with this call)

> *<thinking>The initial image search did not yield a clear match for a plant endemic to southwestern China with a common name resembling an animal feature. The closest matches were for trees like Weeping Elm or cherry trees, which are not consistent with the description in the question. Therefore, I need to refine the search by focusing on plants native to southwestern China that fit the description—specifically, a flowering plant with a common name resembling an animal feature and belonging to a family including genera common in temperate Northern Hemisphere regions (like Fagaceae, which includes oaks, beeches, and chestnuts).*
> *I will now conduct a targeted text search for plants endemic to southwestern China with a common name resembling an animal feature and belonging to the Fagaceae family.</thinking>*

ARGUMENTS

```
{
  "query":  "flowering plant endemic to southwestern China common name resembles animal
feature Fagaceae family",
  "search_type":  "search"
}
```


TOOL OUTPUT


```
Tool `Text_Search_Tool` (tool_call_id=chatcmpl-tool-a448caf97cda639a) returned:
{"ok":  true, "result":  "[Search Results | Type:  search]\nNO RELEVANT RESULTS\n\nWhy each
raw result was rejected:\nAll results describe the Fagaceae family broadly but do not
identify any species endemic to southwestern China with a common name resembling an animal
feature.  None of the results mention such a plant, even though they detail genera, species
counts, and geographic distributions (e.g., Europe, not China).\n\nSuggested query
rewrites:\n- endemic flowering plant in southwestern China with animal-like common name
Fagaceae (rationale:  none of the results address this specific combination)\n- Chinese
endemic plant with animal name in Fagaceae family (rationale:  results focus on general
family traits, not regional endemics with common names)\n\nSuggested next actions:\n- Text_
Search_Tool(query='endemic flowering plant southwestern China animal common name Fagaceae',
search_type=plants)\n- Text_Search_Tool(query='Chinese endemic Fagaceae plant with animal
name', search_type=plants)", "error":  null}
```

Example 9 · **Step 3** Text Search

MODEL (text emitted with this call)

> *<thinking>The initial text search did not yield relevant results. I need to refine the query further to focus on specific plant species endemic to southwestern China that belong to the Fagaceae family and have a common name resembling an animal feature. A strong candidate is *Castanopsis argentea*, commonly known as "silver chestnut" or sometimes colloquially associated with "horse chestnut" due to its fruit resemblance, although "horse chestnut" usually refers to *Aesculus*. Another possibility is *Quercus* spp., but these are generally not described with animal-feature common names.*
> *Given the description in the question, the plant is likely *Castanopsis argentea*, which is endemic to Yunnan, Guizhou, Sichuan, and Guangxi provinces in southwestern China. Let me verify this by searching for "Castanopsis argentea endemic regions China".</thinking>*

ARGUMENTS


```
{
  "query":  "Castanopsis argentea endemic regions China",
  "search_type":  "search"
}
```


TOOL OUTPUT


```
Tool `Text_Search_Tool` (tool_call_id=chatcmpl-tool-8e4049dbe4cbb485) returned:
{"ok":  true, "result":  "[Search Results | Type:  search]\n1.  Castanopsis argentea -
Wikipedia\n Source:  en.wikipedia.org/wiki/Castanopsis_argentea [webpage_url_4]\n It is
mostly found in undisturbed forests of West Java around Mount Gede and Mount Halimun, with
a density of ca.  12 trees/hectare.  It also occurs in national ...\n2.  Ecological
Dynamics of Forest Stands with Castanopsis argentea ...\n Source:  mdpi.com/2073-445X/14/6/
1187 [webpage_url_5]\n This species can be found in primary or old secondary forests,
typically on dry, fertile soils with stoniness at altitudes ranging from 150 to 1750 m
above sea ...\n3.  [PDF] Vegetation community and species association of Castanopsis spp
...\n Source:  smujo.id/biodiv/article/download/9223/5257/51208 [webpage_url_6]\n Previous
studies revealed that.  Castanopsis species were distributed abundantly, especially in
South China and the Indo-China region (Chen et al.  2013; Cheuk
...\n\n[Summary]\nCastanopsis argentea is mostly found in undisturbed forests of West Java
around Mount Gede and Mount Halimun, with a density of ca.  12 trees/hectare, as noted in
its Wikipedia entry.  It can also be found in primary or old secondary forests in China,
typically on dry, fertile soils at altitudes from 150 to 1750 m above sea level.  Previous
studies indicate that Castanopsis species are distributed abundantly in South China and the
Indo-China region, with China hosting 30 endemic species of Castanopsis (Li et al., 2015)."
, "error":  null}
```


HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)


```
Memory update:  registers 3 new webpage URLs in memory (webpage_url_4 to webpage_url_6, from
Text_Search_Tool).
```

Example 9 · **Step 4** Webpage Visit

MODEL (text emitted with this call)

> *<thinking>The search results suggest that while *Castanopsis argentea* is mentioned, its primary distribution is in West Java, Indonesia, not China. However, the third result indicates that Castanopsis species are abundant in South China and the Indo-China region, and China hosts 30 endemic species of Castanopsis. This implies that the plant in question might be a different species within the *Castanopsis* genus endemic to southwestern China.*
> *Let me visit the Wikipedia page for *Castanopsis argentea* to check for any mention of Chinese distribution, and then search for a specific *Castanopsis* species endemic to southwestern China.</thinking>*

ARGUMENTS


```
{
  "urls": ["webpage_url_4"],
  "goal": "Confirm whether Castanopsis argentea is native to any regions in China or if
there are other Castanopsis species endemic to southwestern China."
}
```


TOOL OUTPUT


```
Tool `Webpage_Visit_Tool` (tool_call_id=chatcmpl-tool-9f5dcae3e06d2a9c) returned:
{"ok": true, "result": "The useful information in https://en.wikipedia.org/wiki/
Castanopsis_argentea for user goal Confirm whether Castanopsis argentea is native to any
regions in China or if there are other Castanopsis species endemic to southwestern China.
as follows: \n\nEvidence in page: \nCastanopsis argentea is native to the islands of Java
and Sumatra. Plants of the World Online describes a broader range for this species, from
Assam in eastern India through Myanmar and Thailand, as well as on Sumatra, Java, and
Borneo. There is no mention of China in the text.\n\nSummary: \nCastanopsis argentea is
native to Java and Sumatra, with a broader range including Assam, Myanmar, Thailand, and
Borneo, but not China. The webpage does not mention any Castanopsis species endemic to
southwestern China.\n\n", "error": null}
```


Example 9 · **Step 5** Text Search

MODEL (text emitted with this call)

> *<thinking>Since *Castanopsis argentea* is not native to China, I need to search for a different species within the *Castanopsis* genus that is endemic to southwestern China. Based on the earlier search result mentioning "South China and the Indo-China region" and "30 endemic species of Castanopsis in China, " I will now search for a specific *Castanopsis* species native to southwestern China, particularly one whose common name resembles an animal feature.</thinking>*

ARGUMENTS


```
{
  "query": "Castanopsis species endemic to southwestern China common name animal feature",
  "search_type": "search"
}
```


TOOL OUTPUT


```
Tool `Text_Search_Tool` (tool_call_id=chatcmpl-tool-8ce03e40fd36b2a9) returned:
{"ok": true, "result": "[Search Results | Type: search]\n1. Castanopsis\n Source:
en.wikipedia.org/wiki/Castanopsis [webpage_url_7]\n Castanopsis, commonly called chinquapin
or chinkapin, is a genus of evergreen trees belonging to the beech family, Fagaceae.\n2.
The first report of an 'evergreen Castanopsis type' wood ...\n Source:
onlinelibrary.wiley.com/doi/full/10.1002/gj.3919 [webpage_url_8]\n by D Mantzouka · 2021 ·
Cited by 6 | Yunnan (38 spp.) has the highest Castanopsis species diversity in China, with
11 species endemic to Yunnan, and Indochina has 82 species.\n3. Castanopsis -
AsianFagaceae.com\n Source: asianfagaceae.com/castanopsis [webpage_url_9]\n About half of
the species are found in China, and 30 spp. are endemic (Li et al., 2015), in a wide range
of habitat types. They are often keystone species in ...\n\n[Summary]\nCastanopsis,
commonly called chinquapin or chinkapin, is a genus of evergreen trees in the beech family
Fagaceae. Yunnan province in southwestern China hosts the highest diversity of Castanopsis
species (38 spp.), with 11 species endemic to the region. About half of all Castanopsis
species are found in China, with 30 species endemic to the country, and they often serve as
keystone species in subtropical forests. The genus includes species like Castanopsis
tibetana and Castanopsis chinensis, which are native to China and play important ecological
roles in mountain ecosystems.", "error": null}
```


HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 3 new webpage URLs in memory (webpage_url_7 to webpage_url_9, from
Text_Search_Tool).
```

Example 9 · **Final answer**

MODEL

*<thinking>The search results confirm that the genus *Castanopsis* includes many species endemic to southwestern China, particularly Yunnan province, which has 38 species of *Castanopsis*, including 11 that are endemic. The common name "chinquapin" or "chinkapin" resembles an animal feature (possibly "chipmunk" or "chinchilla"), fitting the description in the question. Additionally, the plant belongs to the Fagaceae family, which includes oaks and beeches—common in temperate regions of the Northern Hemisphere.*
*Therefore, the plant in the image is likely a species of *Castanopsis*, and it is native to southwestern China, primarily Yunnan, but also potentially extending to neighboring provinces like Guizhou, Sichuan, and Guangxi, as suggested by the earlier search results.*
I am now confident in providing the answer.</thinking> <answer>Yunnan, Guizhou, Sichuan, and Guangxi provinces in southwestern China</answer>

What to notice. The most useful part of this trajectory is the self-correction in Steps 3 and 4: the model states a specific hypothesis, looks it up, finds that the species is native to Java and Sumatra rather than China, and discards it. The weakness is the ending: the province list it answers with came from its own earlier guess, and the retrieved pages support only part of it.

Trajectory: `outputs/Qwen3-VL-32B-Instruct/baseagentwithtools/..._20260523_070830_default/BrowseCompVL/full_trajectories/349.json`

END OF EXAMPLE 9

Example 10 Dataset: RealWorldQA Model: Qwen3-VL-32B

Overview (our summary)

Task. In a photograph taken from a car at a city intersection, are there any cars closer to the camera than the nearest crosswalk?
Why it is hard. Judging relative distance along a street from one image is error-prone; without tools the same model answers "yes", pointing at a silver minivan.
How the agent solves it. It estimates metric depth for the whole image, grounds the crosswalk and the cars, and has the coding agent compare the crosswalk's depth with each car's. The crosswalk is 17.4 m away and no car is closer.

INPUT IMAGES



input_image_1

QUESTION (as sent to the model)

```
Question:  Question:  Are there any cars closer to us than the nearest crosswalk?
Options:
A. Yes
B. No
Please select the correct answer from the options above.
```

WITHOUT TOOLS (Qwen3-VL-32B's full response to the same question; source: no-tool run of Qwen3-VL-32B, sandbox/outputs/.../20260522_223842_default)

```
A. Yes

There is a silver minivan on the left side of the street, stopped at the intersection with
its headlights on, which is clearly closer to the viewer than the crosswalk visible on the
right side of the image.  Therefore, there are cars closer to us than the nearest
crosswalk.

Correct answer:  **A. Yes**
```

RESULTS

Qwen3-VL-32B, no tools	A. Yes ✗
Qwen3-VL-32B + OpenVLHarness	**B. No** ✓
Ground truth	B

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 10 · **Step 1** Depth Estimation

MODEL (text emitted with this call)

> *<thinking>First, I need to locate the nearest crosswalk in the image. Then, I will check if there are any cars positioned closer to the camera than that crosswalk. To do this accurately, I'll use the Depth_Estimation_Tool to estimate relative distances and identify which objects (cars and crosswalk) are closer to the viewer.</thinking>*

ARGUMENTS

```
{
  "image_id": "input_image_1",
  "colormap": "inferno"
}
```

TOOL OUTPUT

```
Tool `Depth_Estimation_Tool` (tool_call_id=chatcmpl-tool-ba50db0a90d9afc0) returned:
Depth estimation complete. Shape: [938, 1442], colormap: inferno.
Metric depth in meters available as `depth_meters` (float32 ndarray).
Colorized depth map rendered below; refer to it as tool_generated_image_1.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_1, from Depth_
Estimation_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- depth_result: dict (keys: depth_shape, colormap, size, depth_meters)
```

Example 10 · **Step 2** Visual Grounding

MODEL (text emitted with this call)

> *<thinking>Now I will locate the crosswalk in the image. The crosswalk is visible on the right side of the road, near the bottom of the image. From the depth map (tool_generated_image_1), the crosswalk appears in a region with relatively low depth values (dark purple/blue), indicating it is close to the camera. Next, I need to identify any cars that are even closer than this crosswalk. In the original image, there is a silver car on the left side of the road, just past the stop sign, which appears to be very close to the camera. Its position in the depth map also shows a low depth value, suggesting it is closer than the crosswalk. To confirm this, I will use the Visual_Grounding_Tool to locate both the crosswalk and the cars.</thinking>*

ARGUMENTS

```
{
  "query": "crosswalk",
  "image_id": "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 1 objects for the prompt 'crosswalk'. Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: {'object 1': [788.82, 646.79,
```

```
1000, 744.88]}.  The object numbers correspond to the labeled masks rendered in the
following image, which you can refer to as tool_generated_image_2.
```

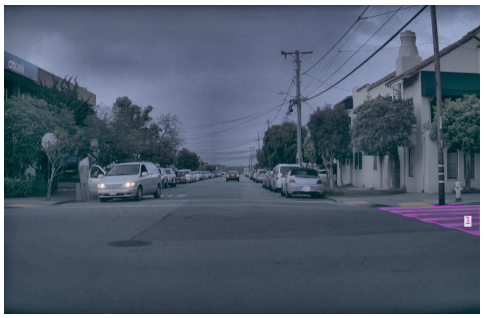


Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_2, from Visual_
Grounding_Tool).
- segment_crosswalk:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 10 · **Step 3** Visual Grounding

MODEL (text emitted with this call)

> *<thinking>Now I will locate the cars in the image using the Visual_Grounding_Tool. I need to identify any cars that are closer to the camera than the crosswalk, which is located on the right side of the road near the bottom. The car on the left side of the road appears to be very close to the camera and may be closer than the crosswalk.</thinking>*

ARGUMENTS

```
{
  "query":  "cars",
  "image_id":  "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 15 objects for the prompt 'cars'.  Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  {'object 1':  [176.53, 513.87,
214.9, 620.67], 'object 2':  [196.14, 505.4, 330.53, 637.25], 'object 3':  [325.42, 525.2,
345.27, 592.08], 'object 4':  [335.01, 525.61, 362.31, 590.18], 'object 5':  [361.83,
532.05, 391.95, 577.99], 'object 6':  [389.67, 539.5, 403.25, 573.49], 'object 7':
[401.02, 535.78, 414.74, 568.96], 'object 8':  [415.52, 537.59, 429.71, 565.0], 'object 9':
[466.41, 534.55, 494.33, 571.55], 'object 10':  [515.88, 533.44, 529.58, 567.8], 'object
11':  [525.95, 530.37, 551.8, 575.87], 'object 12':  [541.83, 534.66, 565.86, 594.67],
'object 13':  [558.22, 512.4, 620.77, 606.17], 'object 14':  [582.0, 524.63, 671.94,
624.84], 'object 15':  [658.99, 527.86, 679.94, 580.83]}.  The object numbers correspond to
the labeled masks rendered in the following image, which you can refer to as tool_generated_
image_3.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_3, from Visual_
Grounding_Tool).
- segment_cars:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 10 · **Step 4** Python Coding Agent

MODEL (text emitted with this call)

> *<thinking>Now I will use the Python_Coding_Agent_Tool to compute the average depth at the center of each detected car and compare it with the average depth of the crosswalk. This will allow me to determine if any cars are closer to the camera than the crosswalk.</thinking>*

ARGUMENTS

```
{
  "objective":  "Compute the average depth at the center of each detected car bounding box
and the average depth at the center of the crosswalk bounding box.  Compare these depths to
determine if any cars are closer to the camera than the crosswalk."
}
```

TOOL OUTPUT

````
Coding agent generated and executed the following Python code:
```python
import numpy as np

# Extract depth map and its dimensions
depth_map = depth_result["depth_meters"]
height, width = depth_result["depth_shape"]

# Extract crosswalk bounding box
crosswalk_box = segment_crosswalk["pred_boxes"][0]
crosswalk_center_x = (crosswalk_box[0] + crosswalk_box[2]) / 2
crosswalk_center_y = (crosswalk_box[1] + crosswalk_box[3]) / 2

# Convert center coordinates to integer (pixel indices)
crosswalk_center_x_int = int(crosswalk_center_x)
crosswalk_center_y_int = int(crosswalk_center_y)

# Get depth at crosswalk center
crosswalk_depth = depth_map[crosswalk_center_y_int, crosswalk_center_x_int]

# Extract car bounding boxes
car_boxes = segment_cars["pred_boxes"]

# Compute average depth at center of each car bounding box
car_centers_depths = []
for box in car_boxes:
    center_x = (box[0] + box[2]) / 2
    center_y = (box[1] + box[3]) / 2
    center_x_int = int(center_x)
    center_y_int = int(center_y)
    car_depth = depth_map[center_y_int, center_x_int]
    car_centers_depths.append(car_depth)

# Compute average depth for all cars
avg_car_depth = np.mean(car_centers_depths)

# Compare:  any car closer than crosswalk?
cars_closer_than_crosswalk = any(car_depth < crosswalk_depth for car_depth in car_centers_
depths)

# Output results
print(f"Crosswalk center depth:  {crosswalk_depth:.2f} meters")
print(f"Average car center depth:  {avg_car_depth:.2f} meters")
print(f"Any car closer to camera than crosswalk?  {'Yes' if cars_closer_than_crosswalk else
'No'}")

# Assign results to variables for result namespace
crosswalk_center_depth = crosswalk_depth
average_car_center_depth = avg_car_depth
any_car_closer_than_crosswalk = cars_closer_than_crosswalk
```
````

```
Execution output:
Crosswalk center depth:  17.40 meters
Average car center depth:  54.14 meters
Any car closer to camera than crosswalk?  No
```

Example 10 · **Final answer**

MODEL

> *<thinking>Based on the depth analysis, the crosswalk is located at an average depth of 17.40 meters from the camera, while the average depth of the cars is 54.14 meters. This indicates that all cars are farther away from the camera than the crosswalk. Therefore, there are no cars closer to us than the nearest crosswalk.</thinking> <answer>B. No</answer>*

What to notice. The question compares distances of two kinds of object along a street, which is easy to misjudge in a single photograph. The harness turns it into arithmetic: depth estimation gives metric depth, grounding gives the crosswalk and every car, and a few lines of code compare them. The detections and the depth map reach the code as data bindings held in memory, `segment_crosswalk`, `segment_cars` and `depth_result`.

Trajectory: `outputs/Qwen3-VL-32B-Instruct/baseagentwithtoolsandcode/..._20260526_095435_default/RealWorldQA/full_trajectories/77.json`

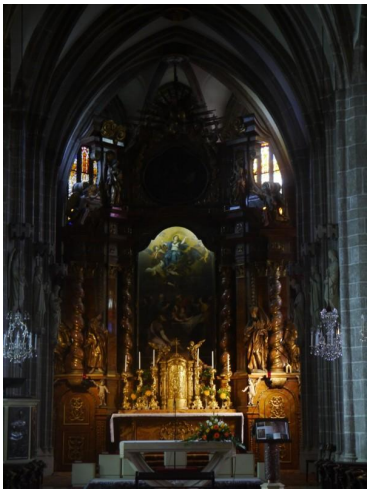
END OF EXAMPLE 10

Example 11 Dataset: SimpleVQA Model: GPT-6 Sol

Overview (our summary)

Task. Name the church in a photograph of a dark Gothic interior with a large baroque altar.
Why it is hard. Many Gothic churches look alike; without tools the model guesses Antwerp Cathedral.
How the agent solves it. Searching the whole photograph and describing it in text return only look-alike churches for eleven calls. The breakthrough is to crop the altarpiece alone and reverse-search the crop, which finds an archive photo of this exact painting; a final text search confirms the church.

INPUT IMAGES



input_image_1

QUESTION (as sent to the model)

```
Question:  What church is depicted in this picture?
```

WITHOUT TOOLS (GPT-6 Sol's full response to the same question)

```
This appears to be the Cathedral of Our Lady in Antwerp, Belgium.
```

RESULTS

GPT-6 Sol, no tools	Cathedral of Our Lady in Antwerp, Belgium ✗
GPT-6 Sol + OpenVLHarness	**St. Augustine's Parish Church in Perchtoldsdorf, Austria** ✓
Ground truth	Pfarrkirche hl. Augustinus, Perchtoldsdorf

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 11 · **Step 1** Image Search

WHAT HAPPENS Reverse image search on the whole photograph. The matches are other churches with similar interiors.

ARGUMENTS

```
{
  "image_id":  "input_image_1"
}
```


TOOL OUTPUT


```
Tool `Image_Search_Tool` (tool_call_id=call_oH6X4bFwHKAz5nlS5Tjs5Fg8) returned:
{"ok":  true, "result":  "[Image Search | Visual Matches]
1.  St Hilda's:  \It was like a Cathedral!" { Shrine of the Relic Image of Our Lady of
Guadalupe
   Source:  relicourladyofguadalupe.wordpress.com/.../st-hildas-it-was-like-a-cathedral
[webpage_url_1]
   Image:  [tool_generated_image_1]
2.  Oh Mary conceived without sin, pray for us who have recourse to Thee!
   Source:  instagram.com/p/DRkWXrTjOOL [webpage_url_2]
   Image:  [tool_generated_image_2]
```


[8 more results not shown (10 in total).]


```
    Image:  [tool_generated_image_12]", "error":  null}
```


HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 12 new images in memory (tool_generated_image_1 to tool_generated_
image_12, from Image_Search_Tool); registers 8 new webpage URLs in memory (webpage_url_1 to
webpage_url_8, from Image_Search_Tool).
```

Example 11 · Step 2 — Webpage Visit

WHAT HAPPENS The agent visits the top match's page to check it; it describes a different church.

ARGUMENTS


```
{
  "urls":  ["webpage_url_1"],
  "goal":  "Determine whether the photograph of dark Gothic church with ornate dark wood
and gold baroque high altar is shown and identify church location."
}
```


TOOL OUTPUT


```
Tool `Webpage_Visit_Tool` (tool_call_id=call_86VJYX7gpuDLXJWAcBH74xsS) returned:
{"ok":  true, "result":  "The useful information in https://
relicourladyofguadalupe.wordpress.com/2017/10/14/st-hildas-it-was-like-a-cathedral/ for
user goal Determine whether the photograph of dark Gothic church with ornate dark wood and
gold baroque high altar is shown and identify church location.  as follows:

Evidence in page:
St Hilda's:  \It was like a Cathedral!" { Shrine of the Relic Image of Our Lady of
Guadalupe

Father Michael Doherty, Parish Priest of St Hilda's Church, Stevenage, was clearly
delighted with the Pilgrimage Visit of the Relic Image.  \It was like being a cathedral for
the weekend!" he exclaimed.  \St Hilda's has become the National Shrine of Our Lady of
Guadalupe!"

He was standing in his beautiful church with its cathedral-sized sanctuary, which was
lavishly decorated with flowers { mainly roses for Our Lady, but also a number of cheerful
sunflowers on the back wall and, of course, with the Relic Image taking pride of place in
the sanctuary.
```


[The remaining 5,465 characters of this output are not shown (6,528 in total).]

Example 11 · Step 3 — Text Search

WHAT HAPPENS A text search that describes the interior in words. The top result is Antwerp Cathedral, the same guess the model makes without tools.

ARGUMENTS


```
{
  "query":  "church interior gothic stone pointed arch dark carved wooden baroque altar
painting virgin mary angels chandeliers",
  "search_type":  "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_C0u5rG9aZlse8h5wvlCYCE2n) returned:
{"ok": true, "result": "[Search Results | Type: images]
1. Cathedral of Our Lady in Antwerp: Rubens, Gothic architecture and UNESCO
   Image: [tool_generated_image_13]
   Source: FromPlaceToPlace.travel [webpage_url_9]
   Size: 768x1024
2. Baroque Altar
   Image: [tool_generated_image_14]
   Source: Instagram [webpage_url_10]
   Size: 640x1136
```

[8 more results not shown (10 in total).]

```
   Size: 800x533", "error": null}
```

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 10 new images in memory (tool_generated_image_13 to tool_generated_
image_22, from Text_Search_Tool); registers 9 new webpage URLs in memory (webpage_url_9 to
webpage_url_17, from Text_Search_Tool).
```

Example 11 · Step 4 Zoom In

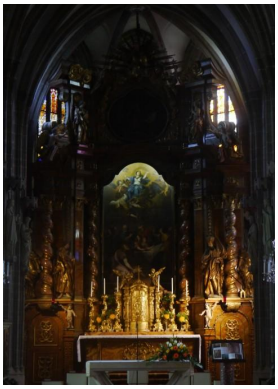
WHAT HAPPENS A wide crop of the nave and altar, used as a new image to search with.

ARGUMENTS

```
{
  "image_id": "input_image_1",
  "coords": [125, 155, 830, 905],
  "padding": 50
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_pjFOkd8Yz6Q1ZeiGYRJcNHfo) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_23.
```



Crop of the nave and altar.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_23, from Zoom_In_Tool).
```

Example 11 · Step 5 Image Search

WHAT HAPPENS Reverse image search on that crop. The matches are still generic Gothic interiors.

ARGUMENTS

```
{
  "image_id": "tool_generated_image_23"
}
```

TOOL OUTPUT

```
Tool `Image_Search_Tool` (tool_call_id=call_RJdprFbLE4EF2X3BcNmtd6GK) returned:
{"ok": true, "result": "[Image Search | Visual Matches]
1. So beautiful Gothic Architecture!
   Source: facebook.com/.../10241119283933448 [webpage_url_18]
   Image: [tool_generated_image_24]
2. This is St. Salvator Church, a Roman Catholic Church in the Klementinum, in Old Town,
Prague, Czech Republic. As early as 1578 - 1581, the Society of Jesus, led by Rector
Giovanni
   Source: reddit.com/.../this_is_st_salvator_church_a_roman_catholic [webpage_url_19]
   Image: [tool_generated_image_25]
```

[8 more results not shown (10 in total).]

```
   Image: [tool_generated_image_34]", "error": null}
```

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 11 new images in memory (tool_generated_image_24 to tool_generated_
image_34, from Image_Search_Tool); registers 9 new webpage URLs in memory (webpage_url_18 to
webpage_url_26, from Image_Search_Tool).
```

Example 11 · Step 6 — Text Search

WHAT HAPPENS A text search guessing a specific cathedral, with German keywords. It surfaces Münster Cathedral.

ARGUMENTS

```
{
  "query": "\"Stephansdom\" \"altar\" \"Maria\" \"barock\" \"Seitenaltar\" schwarze
Säulen",
  "search_type": "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_1zLdigDU9Lxb3Ukttglm3K77) returned:
{"ok": true, "result": "[Search Results | Type: images]
1. wdf - wupper digitale fotografie - Münster, St. Paulus-Dom
   Image: [tool_generated_image_35]
   Source: wupper digitale fotografie [webpage_url_27]
   Size: 1000x1500
2. wdf - wupper digitale fotografie - [] Sakralkunst
   Image: [tool_generated_image_36]
   Source: wupper digitale fotografie [webpage_url_28]
   Size: 764x1021
```

[8 more results not shown (10 in total).]

```
   Size: 764x1021", "error": null}
```

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 10 new images in memory (tool_generated_image_35 to tool_generated_
image_44, from Text_Search_Tool); registers 5 new webpage URLs in memory (webpage_url_27 to
webpage_url_31, from Text_Search_Tool).
```

Example 11 · Step 7 — Image Search

WHAT HAPPENS Reverse image search on one of the returned pictures, to test the Münster lead.

ARGUMENTS

```
{
  "image_id": "tool_generated_image_35"
}
```

TOOL OUTPUT

```
Tool `Image_Search_Tool` (tool_call_id=call_JHAzpfmSFZFnJj1CMWty12ej) returned:
{"ok": true, "result": "[Image Search | Visual Matches]
1. wdf - wupper digitale fotografie - Münster, St. Paulus-Dom
   Source: kdwupper.de/paulusdom-in-münster.html [webpage_url_32]
   Image: [tool_generated_image_35]
2. Rochester NY Chapter | American Guild of Organists
   Source: rocago.org [webpage_url_33]
```

```
    Image:  [tool_generated_image_45]
```


[8 more results not shown (10 in total).]


```
    Image:  [tool_generated_image_54]", "error":  null}
```


HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 10 new images in memory (tool_generated_image_45 to tool_generated_
image_54, from Image_Search_Tool); registers 9 new webpage URLs in memory (webpage_url_32 to
webpage_url_40, from Image_Search_Tool).
```

Example 11 · Step 8 Text Search

WHAT HAPPENS A text search on the Münster lead, which brings up other German cathedrals.

ARGUMENTS

```
{
  "query":  "\"St.  Paulus Dom\" \"altar\" \"Himmelfahrt\" \"barock\"",
  "search_type":  "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_juQSJEm9vDllrbjT4ZcCblRU) returned:
{"ok":  true, "result":  "[Search Results | Type:  images]
1.  Dom zu Eichstätt { Wikipedia
   Image:  [tool_generated_image_55]
   Source:  Wikipedia [webpage_url_41]
   Size:  250x333
2.  Monaco di Baviera - Chiesa dello Spirito Santo (1724-1730), affreschi della navata
dipinti da Cosmas Damian Asam tra il 1724 e il 1730.  Rappresentano Re Davide con l'arpa e
la fondazione dell'Ospedale
   Image:  [tool_generated_image_56]
   Source:  Instagram [webpage_url_42]
   Size:  1440x1920
```

[8 more results not shown (10 in total).]


```
    Size:  764x1146", "error":  null}
```


HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 10 new images in memory (tool_generated_image_55 to tool_generated_
image_64, from Text_Search_Tool); registers 9 new webpage URLs in memory (webpage_url_41 to
webpage_url_49, from Text_Search_Tool).
```

Example 11 · Step 9 Webpage Visit

WHAT HAPPENS The agent visits a page about Münster Cathedral's altars. The descriptions do not match the photograph, and the page summary loops.

ARGUMENTS

```
{
  "urls":  ["webpage_url_32"],
  "goal":  "Identify whether this Münster St Paulus Dom contains a tall dark wooden Baroque
altarpiece with an image of Mary dressed blue above saint kneeling and gold tabernacle, and
which altar it is."
}
```

TOOL OUTPUT

```
Tool `Webpage_Visit_Tool` (tool_call_id=call_10Se3cWeACLTlacvvK4aENmK) returned:
{"ok":  true, "result":  "The useful information in https://www.kdwupper.de/
paulusdom-in-münster.html for user goal Identify whether this Münster St Paulus Dom
contains a tall dark wooden Baroque altarpiece with an image of Mary dressed blue above
saint kneeling and gold tabernacle, and which altar it is.  as follows:

Evidence in page:
{
  "rationale":  "The user is seeking to identify a specific Baroque altarpiece in the St.
Paulus-Dom, Münster, described as tall, dark wooden, with an image of Mary dressed blue
above a kneeling saint and a gold tabernacle.  The webpage content contains detailed
```

```
descriptions of altars and their features, including the Hauptaltar and the
Laurentiusaltar, which match this description.",
```

[The remaining 130,110 characters of this output are not shown (130,916 in total).]

Example 11 · **Step 10** Text Search

WHAT HAPPENS Another descriptive text search; it returns stock photos of altars.

ARGUMENTS

```
{
  "query":  "gothic cathedral baroque altar \"Madonna\" \"kneeling\" dark wood gold
tabernacle",
  "search_type":  "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_vvXGQlwI8M5zkRMwcdjcA3CR) returned:
{"ok":  true, "result":  "[Search Results | Type:  images]
1.  676 Gold Tabernacle Stock Photos - Free & Royalty-Free Stock Photos from Dreamstime
   Image:  [tool_generated_image_65]
   Source:  Dreamstime [webpage_url_50]
   Size:  600x900
2.  676 Gold Tabernacle Stock Photos - Free & Royalty-Free Stock Photos from Dreamstime
   Image:  [tool_generated_image_66]
   Source:  Dreamstime [webpage_url_50]
   Size:  600x900
```

[8 more results not shown (10 in total).]

```
   Size:  600x900", "error":  null}
```

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 10 new images in memory (tool_generated_image_65 to tool_generated_
image_74, from Text_Search_Tool); registers 8 new webpage URLs in memory (webpage_url_50 to
webpage_url_57, from Text_Search_Tool).
```

Example 11 · **Step 11** Text Search

WHAT HAPPENS A German-language descriptive search; still no match.

ARGUMENTS

```
{
  "query":  "\"gotische\" \"Kirche\" \"barocker\" \"Hochaltar\" \"Rosenkranz\" \"
Tabernakel\"",
  "search_type":  "images"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_W4yiz0YOUUZ9djQcKEyENQj6) returned:
{"ok":  true, "result":  "[Search Results | Type:  images]
1.  verlag-st-peter.at - Die Kirchen von Hopfgarten im Brixental - Pfarrkirche:  Rundgang
Altäre
   Image:  [tool_generated_image_75]
   Source:  Kirchenführer aus dem Verlag St.  Peter [webpage_url_58]
   Size:  400x600
2.  Barockheiligtum mit hochaltar -Fotos und -Bildmaterial in hoher Auflösung { Alamy
   Image:  [tool_generated_image_76]
   Source:  Alamy [webpage_url_59]
   Size:  1300x956
```

[8 more results not shown (10 in total).]

```
   Size:  400x600", "error":  null}
```

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 10 new images in memory (tool_generated_image_75 to tool_generated_
image_84, from Text_Search_Tool); registers 2 new webpage URLs in memory (webpage_url_58 to
webpage_url_59, from Text_Search_Tool).
```

Example 11 · **Step 12** Zoom In

WHAT HAPPENS The key step: a tight crop of the altarpiece alone, the one detail unique to this church.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [360, 440, 590, 713],
  "padding":  50
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_o1GC4x3B6pWmL7FYCQuhlMR8) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_85.
```



Crop of the altarpiece alone.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_85, from Zoom_In_Tool).
```

Example 11 · **Step 13** Image Search

WHAT HAPPENS Reverse image search on the altarpiece crop. The first result is a Wikimedia Commons photo of the high-altar painting of the parish church in Perchtoldsdorf.

ARGUMENTS

```
{
  "image_id":  "tool_generated_image_85"
}
```

TOOL OUTPUT

```
Tool `Image_Search_Tool` (tool_call_id=call_hPFqkdstj5TH5Bkj2yYmJHFE) returned:
{"ok":  true, "result":  "[Image Search | Visual Matches]
1.  File:Perchtoldsdorf Pfarrkirche Hochaltarbild 01.jpg - Wikimedia Commons
   Source:  commons.wikimedia.org/wiki/File:Perchtoldsdorf_Pfarrkirche_Hochaltarbild_01.jpg
[tool_generated_image_86]
   Image:  [tool_generated_image_87]
2.  Solum Church - Wikipedia
   Source:  en.wikipedia.org/wiki/Solum_Church [webpage_url_60]
   Image:  [tool_generated_image_88]
```

[8 more results not shown (10 in total).]

```
   Image:  [tool_generated_image_97]", "error":  null}
```

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 12 new images in memory (tool_generated_image_86 to tool_generated_
image_97, from Image_Search_Tool); registers 8 new webpage URLs in memory (webpage_url_60 to
webpage_url_67, from Image_Search_Tool).
```

Example 11 · **Step 14** Text Search

WHAT HAPPENS A text search on that lead confirms it: the parish church of St. Augustine in Perchtoldsdorf has a high altar with an Assumption of Mary.

ARGUMENTS

```
{
  "query": "Perchtoldsdorf Pfarrkirche Hochaltar Kirche Mariä Himmelfahrt altar",
  "search_type": "search"
}
```

TOOL OUTPUT

```
Tool `Text_Search_Tool` (tool_call_id=call_8aq24tQS1GhdDHIDdzwjZ5Q6) returned:
{"ok": true, "result": "[Search Results | Type: search]
1. Pfarrkirche St. Augustinus und Pfarrhof
   Source: perchtoldsdorf.at/Pfarrkirche_St_Augustinus_und_Pfarrhof [webpage_url_68]
   von Perchtoldsdorf im Jahre 1217 zur Pfarrkirche. Der Hochaltar mit tordierten Säulen.
Hochaltar mit dem Hauptbild ,,Mariä Himmelfahrt\ und dem hl. Augustinus ...
2. Pfarrkirche | röm.-kath. Pfarre Perchtoldsdorf
   Source: pfarre-perchtoldsdorf.at/unsere-pfarre/pfarrkirche [webpage_url_69]
   Neuigkeiten, bevorstehende Veranstaltungen und Informationen zu Taufen, Trauungen und
weiteren kirchlichen Anlässen der röm. kath. Pfarre Perchtoldsdorf.
```

[8 more results not shown (10 in total).]

```
   Die römisch-katholische Pfarrkirche Perchtoldsdorf steht am Marktplatz in
Perchtoldsdorf, Bezirk Mödling, Niederösterreich, im Bereich der ehemaligen Burg.", "error"
: null}
```

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 9 new webpage URLs in memory (webpage_url_68 to webpage_url_76,
from Text_Search_Tool).
```

HARNESS

```
Please provide your best final answer now if enough information is available.
```

Example 11 · **Final answer**

WHAT HAPPENS After the harness asks for a final answer, the agent names St. Augustine's Parish Church in Perchtoldsdorf, Austria, which the judge accepts.

MODEL

```
<answer>St. Augustine's Parish Church in Perchtoldsdorf, Austria.</answer>
```

What to notice. Searching the whole photograph returns look-alike Gothic interiors, and eleven calls of text search, page visits and a wide crop do not narrow them down; the model tries Antwerp, Münster and Eichstätt along the way. Cropping the altarpiece (Step 12) turns the reverse image search into a specific one: its first result is an archive photo of this exact altar painting (Step 13). A text search on that lead then confirms the church (Step 14). Crop, reverse image search and text verification come from different tool families, composed through images and web pages the harness keeps addressable in memory.

Trajectory:
`outputs/gpt-6-sol/apiagentwithtools/..._sw13_100_tools/SimpleVQA/full_trajectories/1269.json`

END OF EXAMPLE 11

Example 12 Dataset: V* Model: GPT-6 Sol

Overview (our summary)

Task. In a large landscape photograph of a hilltop town, say whether a woman with a backpack is to the left or right of a person wearing a beanie.
Why it is hard. The two people stand near a bus on a road far below the town and are only a few pixels tall; without tools the model answers "right".
How the agent solves it. Grounding on the full image cannot separate the two people, so the agent zooms in three times, each time closer, and grounds both people again inside the final crop, where they are large enough to tell apart.

INPUT IMAGES



input_image_1

QUESTION (as sent to the model)

```
Question:  Question:  Is the woman with a backpack on the left or right side of the person
with a beanie?
Options:
A. right
B. left
Please select the correct answer from the options above.
```

WITHOUT TOOLS (GPT-6 Sol's full response to the same question)

```
A. right
```

RESULTS

GPT-6 Sol, no tools	A. right ✗
GPT-6 Sol + OpenVLHarness	**B. left** ✓
Ground truth	B

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 12 · **Step 1** Visual Grounding

WHAT HAPPENS Grounding "person wearing beanie" on the full image returns one tiny box.

ARGUMENTS

```
{
  "query": "person wearing beanie",
  "image_id": "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 1 objects for the prompt 'person wearing beanie'. Their
corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: ```json
[
    {"bbox_2d": [369, 830, 381, 853], "label": "person wearing beanie"}
]```. The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_1.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_1, from Visual_
Grounding_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- segment_person_wearing_beanie: dict (keys: pred_boxes, pred_scores, pred_masks, num_masks,
orig_img_h, orig_img_w)
```

Example 12 · **Step 2** Visual Grounding

WHAT HAPPENS Grounding "woman with backpack" returns almost the same tiny box, so at this scale the two people cannot be told apart.

ARGUMENTS

```
{
  "query": "woman with backpack",
  "image_id": "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 1 objects for the prompt 'woman with backpack'. Their
corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: ```json
[
    {"bbox_2d": [368, 831, 381, 853], "label": "woman with backpack"}
]```. The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_2.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_2, from Visual_
Grounding_Tool).
- segment_woman_with_backpack:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks,
orig_img_h, orig_img_w)
```

Example 12 · **Step 3** Zoom In

WHAT HAPPENS A first zoom on the region around that box.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [350, 810, 410, 875],
  "padding":  100
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_48KspHqeWAyb0sVqvUkH75he) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_3.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_3, from Zoom_In_Tool).
```

Example 12 · **Step 4** Zoom In

WHAT HAPPENS A second zoom on a nearby region below the bus.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [500, 810, 605, 920],
  "padding":  100
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_K9cLu6IVuAX39EqWkYJa6WuQ) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_4.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_4, from Zoom_In_Tool).
```

Example 12 · **Step 5** Zoom In

WHAT HAPPENS A third, much tighter zoom on the two figures. This crop is the evidence for the answer.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "coords":  [508, 862, 538, 895],
  "padding":  50
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_u8svfarDLYuRGK81UTyQojba) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_5.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_5, from Zoom_In_Tool).
```

Example 12 · **Step 6** Visual Grounding

WHAT HAPPENS Grounding “woman with backpack” inside the crop now returns a clear box for her.

ARGUMENTS

```
{
  "query":  "woman with backpack",
  "image_id":  "tool_generated_image_5"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 1 objects for the prompt 'woman with backpack'.  Their
corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [419, 447, 511, 569], "label":  "woman with backpack"}
]```.  The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_6.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_6, from Visual_
Grounding_Tool).
- segment_woman_with_backpack_2:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks,
orig_img_h, orig_img_w)
```

Example 12 · **Step 7** Visual Grounding

> **WHAT HAPPENS** Grounding "person with beanie" in the crop returns two boxes: the woman's own box again, and a second person to her right.

ARGUMENTS

```
{
  "query":  "person with beanie",
  "image_id":  "tool_generated_image_5"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 2 objects for the prompt 'person with beanie'.  Their
corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [419, 446, 511, 569], "label":  "person with beanie"},
    {"bbox_2d":  [472, 435, 527, 560], "label":  "person with beanie"}
]```.  The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_7.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_7, from Visual_
Grounding_Tool).
- segment_person_with_beanie:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks,
orig_img_h, orig_img_w)
```

Example 12 · **Final answer**

> **WHAT HAPPENS** The agent concludes that the woman is on the left of the person with the beanie, option B. The beanie query also returned the woman's own box, so this last step still relies on the model's reading of the crop.

MODEL

```
<answer>B. left</answer>
```

What to notice. The two people are a few pixels tall in the full image. Grounding them there returns almost the same tiny box for both queries (Steps 1 and 2), so it cannot separate them. The model zooms three times, each time closer, and grounds again on the final crop (Steps 6 and 7). There the woman with a backpack gets her own box, and the beanie query returns a second person to her right, so the model answers that she is on the left. The beanie query also returns the woman's box, so this last step still needs the model's judgement.

Trajectory:
`outputs/gpt-6-sol/apiagentwithtools/..._sw13_100_tools/VStarBench/full_trajectories/162.json`

END OF EXAMPLE 12

Example 13 Dataset: MMSI-Bench Model: GPT-6 Sol

Overview (our summary)

Task. Seven photographs show one living room from different positions. Of four statements about the room, choose the correct one; two of them give the number of throw pillows, 16 or 12, counting partly covered ones.
Why it is hard. The pillows are spread across several views, some appear in more than one view, and some are partly hidden; without tools the model picks 12.
How the agent solves it. It grounds "throw pillows" in each of the four views that show seating, zooms on both halves of the main sofa, and chooses the statement with 16 pillows.

INPUT IMAGES


input_image_1


input_image_2


input_image_3


input_image_4


input_image_5


input_image_6


input_image_7

QUESTION (as sent to the model)

```
Question:  The kitchen is located to the north relative to the sofa.  Which of the
following statements is correct?
Options:  A: The mirror is to the southwest of the refrigerator, B: The radius of the vase
in Figure 2 is larger than the radius of the white lampshade of the table lamp in Figure 5,
C: There are a total of 16 throw pillows in the picture (including those partially covered
by other objects), D: There are a total of 12 throw pillows in the picture (including those
partially covered by other objects),
Answer with the option's letter from the given choices directly.  Enclose the option's
letter within ``.
```

WITHOUT TOOLS (GPT-6 Sol's full response to the same question)

```
`D`
```

RESULTS

GPT-6 Sol, no tools	D (12 throw pillows) ✗
GPT-6 Sol + OpenVLHarness	**C (16 throw pillows)** ✓
Ground truth	C

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
- input_image_2:  user_input
```

```
- input_image_3: user_input
- input_image_4: user_input
- input_image_5: user_input
- input_image_6: user_input
- input_image_7: user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 13 · **Step 1** Visual Grounding

WHAT HAPPENS Grounding in view 5, which shows the main sofa, returns 11 pillows.

ARGUMENTS

```
{
  "query": "throw pillows",
  "image_id": "input_image_5"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 11 objects for the prompt 'throw pillows'. Their
corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: ```json
[
    {"bbox_2d": [0, 684, 113, 866], "label": "throw pillows"},
    {"bbox_2d": [55, 717, 159, 861], "label": "throw pillows"},
    {"bbox_2d": [180, 677, 308, 825], "label": "throw pillows"},
    {"bbox_2d": [326, 709, 428, 830], "label": "throw pillows"},
    {"bbox_2d": [352, 664, 457, 817], "label": "throw pillows"},
    {"bbox_2d": [592, 787, 715, 908], "label": "throw pillows"},
    {"bbox_2d": [631, 707, 772, 841], "label": "throw pillows"},
    {"bbox_2d": [690, 819, 831, 1000], "label": "throw pillows"},
    {"bbox_2d": [755, 738, 898, 993], "label": "throw pillows"},
    {"bbox_2d": [841, 905, 937, 1000], "label": "throw pillows"},
    {"bbox_2d": [930, 816, 1000, 1000], "label": "throw pillows"}
]```. The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_1.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_1, from Visual_
Grounding_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- segment_throw_pillows: dict (keys: pred_boxes, pred_scores, pred_masks, num_masks, orig_
img_h, orig_img_w)
```

Example 13 · **Step 2** Visual Grounding

WHAT HAPPENS Grounding in view 3 returns 3 pillows.

ARGUMENTS

```
{
  "query": "throw pillows",
  "image_id": "input_image_3"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 3 objects for the prompt 'throw pillows'. Their
corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: ```json
[
    {"bbox_2d": [814, 671, 883, 806], "label": "throw pillows"},
    {"bbox_2d": [939, 663, 974, 738], "label": "throw pillows"},
    {"bbox_2d": [963, 669, 1000, 748], "label": "throw pillows"}
]```. The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_2.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_2, from Visual_
Grounding_Tool).
- segment_throw_pillows_2: dict (keys: pred_boxes, pred_scores, pred_masks, num_masks, orig_
img_h, orig_img_w)
```

Example 13 · **Step 3** Visual Grounding

WHAT HAPPENS Grounding in view 4 returns 3 pillows.

ARGUMENTS

```
{
  "query": "throw pillows",
  "image_id": "input_image_4"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 3 objects for the prompt 'throw pillows'. Their
corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: ```json
[
    {"bbox_2d": [0, 658, 46, 729], "label": "throw pillows"},
    {"bbox_2d": [144, 881, 277, 1000], "label": "throw pillows"},
    {"bbox_2d": [963, 724, 1000, 852], "label": "throw pillows"}
]```. The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_3.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_3, from Visual_
Grounding_Tool).
- segment_throw_pillows_3:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_
img_h, orig_img_w)
```

Example 13 · **Step 4** Visual Grounding

WHAT HAPPENS Grounding in view 6 returns 1 pillow.

ARGUMENTS

```
{
  "query":  "throw pillows",
  "image_id":  "input_image_6"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 1 objects for the prompt 'throw pillows'.  Their
corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are:  ```json
[
    {"bbox_2d":  [0, 799, 102, 1000], "label":  "throw pillows"}
]```.  The object numbers correspond to the labeled masks rendered in the following image,
which you can refer to as tool_generated_image_4.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_4, from Visual_
Grounding_Tool).
- segment_throw_pillows_4:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_
img_h, orig_img_w)
```

Example 13 · **Step 5** Zoom In

WHAT HAPPENS A zoom on the lower right of view 5, where pillows on the sofa overlap.

ARGUMENTS

```
{
  "image_id":  "input_image_5",
  "coords":  [530, 660, 1000, 1000],
  "padding":  50
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_MTJTBg06TkGQbxJlDKAYPR3r) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_5.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_5, from Zoom_In_Tool).
```

Example 13 · **Step 6** Zoom In

WHAT HAPPENS A zoom on the lower left of view 5.

ARGUMENTS

```
{
  "image_id":  "input_image_5",
  "coords":  [0, 650, 530, 1000],
  "padding":  50
}
```

TOOL OUTPUT

```
Tool `Zoom_In_Tool` (tool_call_id=call_Ybk2IO4dyyOZDI6yeGGdwGt8) returned:
Zoomed in image rendered below; refer to it as tool_generated_image_6.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_6, from Zoom_In_Tool).
```

Example 13 · **Final answer**

WHAT HAPPENS The grounding calls found 18 pillows in total, but some are the same pillow seen from two views. The agent answers C, 16 pillows, which is correct; it does not write out which detections it treated as duplicates.

MODEL

```
<answer>`C`</answer>
```

What to notice. Of the four options, two are pillow counts, so the question reduces to counting throw pillows across seven views of one room, including partly covered ones. The model grounds "throw pillows" in the four views that show seating (11, 3, 3 and 1 detections, 18 in total), then zooms on the two halves of the sofa in view 5. It answers 16, which requires recognizing that some pillows appear in more than one view. That cross-view deduplication is implicit: the model does not write it out.

Trajectory: `outputs/gpt-6-sol/apiagentwithtools/..._sw13_100_tools/MMSIBench_wo_circular/full_trajectories/267.json`

END OF EXAMPLE 13

Example 14 Dataset: MMSI-Bench Model: Qwen3-VL-32B

Overview (our summary)

Task. Two photographs show two rooms of a historic house: a parlour with a piano, and a dining room. Which is longer, the piano or the table under the mirror? Without tools the same model answers D, that the piano is longer.
How the agent approaches it. It grounds both objects, estimates metric depth in both photos, and has the coding agent convert each box to a real-world length.
Caveat. The answer is correct, but the evidence does not support it. The mirror hangs above the fireplace, so the "table under the mirror" is the mantel shelf. Grounding returns it as object 3, but the agent chooses the round dining table (object 2) instead, and concludes that the table, at 2.77 m, is longer than the piano, at 2.21 m. This example shows the harness's tool chain, not a sound comparison.

INPUT IMAGES



input_image_1



input_image_2

QUESTION (as sent to the model)

```
Question:  Which is longer, the piano or the table under the mirror?
Options:  A: The table under the mirror is longer, B: Sometimes the table under the mirror
is longer, sometimes the piano is longer, C: They are the same length, D: The piano is
longer,
Answer with the option's letter from the given choices directly.  Enclose the option's
letter within ``.
```

RESULTS (no-tool answer from a separate run of the same model, final answer only)

Qwen3-VL-32B, no tools	D (the piano is longer) ✗
Qwen3-VL-32B + OpenVLHarness	**A (the table under the mirror is longer)** ✓
Ground truth	A

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
- input_image_2:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 14 · **Step 1** Visual Grounding

MODEL (text emitted with this call)

> *<thinking>To determine which is longer—the piano or the table under the mirror—I need to compare their lengths. The piano is visible in input_image_1, and the table under the mirror is visible in input_image_2. Since they are in different images, I need to use a tool to estimate their lengths relative to each other. I will first ground both objects in their respective images to get bounding boxes, then use a depth estimation tool to get relative scales, and finally compare them using Python_Coding_Agent_Tool.</thinking>*

ARGUMENTS

```
{
  "query": "piano",
  "image_id": "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 1 objects for the prompt 'piano'. Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: {'object 1': [503.72, 477.32,
787.18, 999.63]}. The object numbers correspond to the labeled masks rendered in the
following image, which you can refer to as tool_generated_image_1.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_1, from Visual_
Grounding_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- segment_piano: dict (keys: pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 14 · **Step 2** Visual Grounding

ARGUMENTS

```
{
  "query": "table under the mirror",
  "image_id": "input_image_2"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 3 objects for the prompt 'table under the mirror'. Their
corresponding bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: {'object 1':
[2.6, 762.44, 205.84, 901.39], 'object 2': [38.62, 823.36, 556.79, 1000], 'object 3':
[294.14, 573.73, 587.94, 865.76]}. The object numbers correspond to the labeled masks
rendered in the following image, which you can refer to as tool_generated_image_2.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_2, from Visual_
Grounding_Tool).
- segment_table_under_the_mirror:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_
masks, orig_img_h, orig_img_w)
```

Example 14 · **Step 3** Depth Estimation

MODEL (text emitted with this call)

> *<thinking>From the grounding results, I have identified the piano in input_image_1 with bounding box [503.72, 477.32, 787.18, 999.63]. In input_image_2, there are three objects grounded under the prompt 'table under the mirror': a small side table (object 1), a round dining table (object 2), and a fireplace mantel (object 3). The most relevant object for comparison is the round dining table (object 2) with bounding box [38.62, 823.36, 556.79, 1000].*
> *To compare their lengths accurately, I need to account for perspective and depth. I will estimate the depth map for both images to get relative depth values at the centers of the piano and the dining table, then compute their projected lengths using the bounding box dimensions and depth information.</thinking>*

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "colormap":  "grayscale"
}
```

TOOL OUTPUT

```
Tool `Depth_Estimation_Tool` (tool_call_id=chatcmpl-tool-a16443994cabdb5b) returned:
Depth estimation complete.  Shape:  [1022, 1274], colormap:  grayscale.
Metric depth in meters available as `depth_meters` (float32 ndarray).
Colorized depth map rendered below; refer to it as tool_generated_image_3.
```

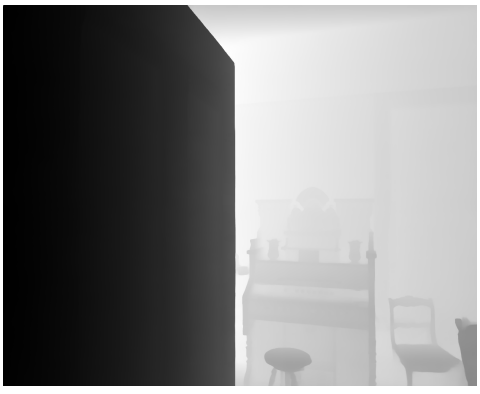


Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_3, from Depth_
Estimation_Tool).
- depth_result: dict (keys: depth_shape, colormap, size, depth_meters)
```

Example 14 · **Step 4** Depth Estimation

ARGUMENTS

```
{
  "image_id": "input_image_2",
  "colormap": "grayscale"
}
```

TOOL OUTPUT

```
Tool `Depth_Estimation_Tool` (tool_call_id=chatcmpl-tool-928a1ad88a9f8106) returned:
Depth estimation complete. Shape: [1022, 1274], colormap: grayscale.
Metric depth in meters available as `depth_meters` (float32 ndarray).
Colorized depth map rendered below; refer to it as tool_generated_image_4.
```



Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_4, from Depth_
Estimation_Tool).
- depth_result_2: dict (keys: depth_shape, colormap, size, depth_meters)
```

Example 14 · **Step 5** Python Coding Agent

MODEL (text emitted with this call)

> *<thinking>Now I have depth maps for both images. I will use the Python_Coding_Agent_Tool to compute the actual lengths of the piano and the dining table (object 2) by combining their bounding box dimensions with the depth information at their centers. This will allow me to compare their real-world lengths accurately.</thinking>*

ARGUMENTS

```
{
  "objective": "Compute the real-world length of the piano from input_image_1 and the
dining table (object 2) from input_image_2 using their bounding boxes and depth values at
their centers. The piano's bounding box is [503.72, 477.32, 787.18, 999.63] and the dining
table's bounding box is [38.62, 823.36, 556.79, 1000]. Use the depth maps from the Depth_
Estimation_Tool to get the depth at the center of each object. Then, calculate the
projected length using the formula: real_length = (bbox_width / image_width) * depth * 2
(assuming a rough perspective correction factor). Compare the two lengths and determine
which is longer."
}
```

TOOL OUTPUT

Coding agent generated and executed the following Python code:

```python
import numpy as np

# Given bounding boxes (from the objective)
piano_bbox = [503.72, 477.32, 787.18, 999.63] # [x1, y1, x2, y2]
table_bbox = [38.62, 823.36, 556.79, 1000] # [x1, y1, x2, y2]

# Image dimensions (from segment_piano and depth_result)
image_width = segment_piano["orig_img_w"] # 1280
image_height = segment_piano["orig_img_h"] # 1024

# Depth maps
depth_map_1 = depth_result["depth_meters"] # for piano (input_image_1)
depth_map_2 = depth_result_2["depth_meters"] # for table (input_image_2)

# Compute center coordinates for each object
piano_center_x = (piano_bbox[0] + piano_bbox[2]) / 2
piano_center_y = (piano_bbox[1] + piano_bbox[3]) / 2
table_center_x = (table_bbox[0] + table_bbox[2]) / 2
table_center_y = (table_bbox[1] + table_bbox[3]) / 2

# Convert center coordinates to integer for depth map indexing
piano_center_x_int = int(piano_center_x)
piano_center_y_int = int(piano_center_y)
table_center_x_int = int(table_center_x)
table_center_y_int = int(table_center_y)

# Get depth at center points (clip to bounds if needed)
# Ensure coordinates are within depth map bounds
piano_center_x_int = max(0, min(piano_center_x_int, depth_map_1.shape[1] - 1))
piano_center_y_int = max(0, min(piano_center_y_int, depth_map_1.shape[0] - 1))
table_center_x_int = max(0, min(table_center_x_int, depth_map_2.shape[1] - 1))
table_center_y_int = max(0, min(table_center_y_int, depth_map_2.shape[0] - 1))

piano_depth = depth_map_1[piano_center_y_int, piano_center_x_int]
table_depth = depth_map_2[table_center_y_int, table_center_x_int]

# Compute projected real-world lengths
# Formula:  real_length = (bbox_width / image_width) * depth * 2
piano_width = piano_bbox[2] - piano_bbox[0]
table_width = table_bbox[2] - table_bbox[0]

piano_real_length = (piano_width / image_width) * piano_depth * 2
table_real_length = (table_width / image_width) * table_depth * 2

# Compare and output results
print(f"Piano real-world length:  {piano_real_length:.4f} meters")
print(f"Dining table real-world length:  {table_real_length:.4f} meters")

if piano_real_length > table_real_length:
    longer_object = "piano"
else:
    longer_object = "dining table"

print(f"The {longer_object} is longer.")

# Assign results to variables for result namespace
piano_real_length = piano_real_length
table_real_length = table_real_length
longer_object = longer_object
```

Execution output:

```
Piano real-world length:  2.2109 meters
Dining table real-world length:  2.7731 meters
The dining table is longer.
```

Example 14 · **Final answer**

MODEL

> *<thinking>The Python_Coding_Agent_Tool computed the real-world lengths of the piano and the dining table. The piano is approximately 2.21 meters long, while the dining table (the round table under the mirror) is approximately 2.77 meters long. Therefore, the table under the mirror is longer than the piano.</thinking> <answer>A</answer>*

What to notice. The model composes grounding, depth and code to turn a comparison between two photographs into a metric one, and it states the plan before the first call. The example also shows a failure mode the harness does not prevent: the model picks the wrong object among the grounding results, and nothing later in the chain checks that choice.

Trajectory: `outputs/Qwen3-VL-32B-Instruct/baseagentwithtoolsandcode/..._20260526_095435_default/MMSIBench_wo_circular/full_trajectories/7.json`

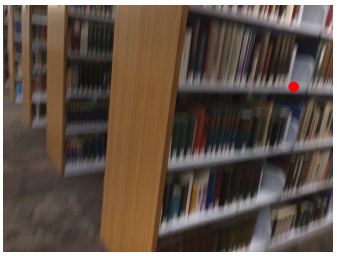
END OF EXAMPLE 14

Example 15 Dataset: SparBench Model: GPT-6 Sol

Overview (our summary)

Task. Three photographs of a library are taken from nearby positions. A red point marks a bookshelf in the first photo and a blue point marks another bookshelf in the third. Give the 3D distance between them in metres.
Why it is hard. The two points never appear in the same photo, so the distance cannot be judged from any single view; without tools the model answers 4.0 m.
How the agent solves it. It recovers where each photo was taken from, measures metric depth at each point, and has the coding agent place both points in one 3D frame and measure between them: 2.0 m, against a ground truth of 1.9 m.

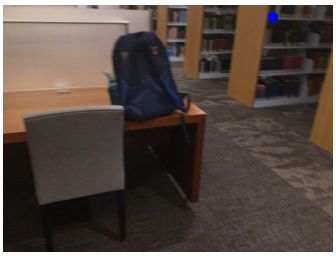
INPUT IMAGES



input_image_1. The red point marks the first bookshelf.



input_image_2



input_image_3. The blue point marks the second bookshelf.

QUESTION (as sent to the model)

```
Question:
State the Euclidean distance in meters from the center of bookshelf (red point) to the
center of bookshelf (blue point).  Calculate or judge based on the 3D center points of
these objects.  Provide only one number for your response.
Please answer the question using a single word or phrase.
```

WITHOUT TOOLS (GPT-6 Sol's full response to the same question)

```
4.0
```

RESULTS (MRA: SparBench's mean relative accuracy over thresholds 0.50–0.95)

	answer	MRA
GPT-6 Sol, no tools	4.0 m	0.00 ✗
GPT-6 Sol + OpenVLHarness	**2.0 m**	**0.90** ✓
Ground truth	1.9 m	

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
- input_image_2:  user_input
- input_image_3:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 15 · **Step 1** Camera Trajectory

WHAT HAPPENS Camera trajectory estimates each camera's position and orientation relative to the first photo. The third photo was taken about 1.5 m to the left and 1.65 m behind the first. The full camera parameters are kept as a data binding for code.

ARGUMENTS

```
{
  "image_ids":  ["input_image_1", "input_image_2", "input_image_3"]
}
```

TOOL OUTPUT

```
Camera trajectory summary:
- input_image_1:  reference camera and reference view.
- input_image_2:
    camera movement:  salient direction:  none (all movements < 1 meter); detailed:  right
(+0.01), down (+0.16), backward (-0.08)
    camera rotation:  salient direction:  left (yaw -27.4°), down (pitch +19.8°); detailed:
left (yaw -27.4°), down (pitch +19.8°)
- input_image_3:
    camera movement:  salient direction:  left (-1.49), backward (-1.65); detailed:  left
(-1.49), down (+0.27), backward (-1.65)
    camera rotation:  salient direction:  left (yaw -12.1°); detailed:  left (yaw -12.1°),
down (pitch +6.1°)
If no salient movement or rotation is reported, use the magnitude of each axis to determine
the most possible movement/rotation direction.

Bird's-eye view of the camera trajectory on input_image_1's X-Z plane is rendered in the
following image - refer as tool_generated_image_1 (dots = camera positions, forward-pointing
arrows = facing direction).

All vectors are relative to input_image_1's camera frame, using the coordinate system:  +X =
right, +Y = down, +Z = forward.  With this convention, positive yaw (rotation around Y
axis) = right and positive pitch (rotation around X axis) = down.  If the original question
assumes a different coordinate system (e.g.  with +Y = up and -Z = forward, which flips the
yaw/pitch signs), convert before answering.
```



Bird's-eye view of the three camera positions in input_image_1's frame.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_1, from Camera_
Trajectory_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- camera_trajectory_result:  dict (keys:  intrinsics, extrinsics, image_ids, num_images,
cameras, coordinate_system, reference_image_id)
```

Example 15 · **Step 2** **Depth Estimation**

WHAT HAPPENS Metric depth for the first photo, where the red point is.

ARGUMENTS

```
{
  "image_id":  "input_image_1",
  "colormap":  "inferno"
}
```

TOOL OUTPUT

```
Tool `Depth_Estimation_Tool` (tool_call_id=call_qrfQNTtmO9TNFVBT4xdZiRyL) returned:
Depth estimation complete.  Shape:  [966, 1302], colormap:  inferno.
Metric depth in meters available as `depth_meters` (float32 ndarray).
Colorized depth map rendered below; refer to it as tool_generated_image_2.
```



Colorized depth map of input_image_1.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_2, from Depth_
Estimation_Tool).
- depth_result:  dict (keys:  depth_shape, colormap, size, depth_meters)
```

Example 15 · **Step 3** Depth Estimation

WHAT HAPPENS Metric depth for the third photo, where the blue point is.

ARGUMENTS

```
{
  "image_id":  "input_image_3",
  "colormap":  "inferno"
}
```

TOOL OUTPUT

```
Tool `Depth_Estimation_Tool` (tool_call_id=call_YwF06eDGu6x1VabL7jtqFml7) returned:
Depth estimation complete.  Shape:  [966, 1302], colormap:  inferno.
Metric depth in meters available as `depth_meters` (float32 ndarray).
Colorized depth map rendered below; refer to it as tool_generated_image_3.
```



Colorized depth map of input_image_3.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_3, from Depth_
Estimation_Tool).
- depth_result_2:  dict (keys:  depth_shape, colormap, size, depth_meters)
```

Example 15 · **Step 4** Python Coding Agent

WHAT HAPPENS The agent gives the coding agent the pixel positions of the two points, which it read off the images itself. The code reads the depth at each point (2.15 m and 4.08 m), turns each point into 3D coordinates using its camera's parameters, and measures the distance between them: 2.0076 m.

ARGUMENTS

```
{
  "objective":  "Estimate Euclidean distance between red dot at pixel (1141,320) in first
1302x966 image and blue dot at pixel (1060,47) in third image using respective estimated
depth maps, camera intrinsics and world-to-camera extrinsics.  Print intrinsics, camera
transforms, depth local median at each point, derived 3D world coordinates and distance.
Be careful depth_result is first image and depth_result_2 is third."
}
```

TOOL OUTPUT

````
Coding agent generated and executed the following Python code:
```python
import numpy as np

# Extract relevant data
image_ids = camera_trajectory_result["image_ids"]
num_images = camera_trajectory_result["num_images"]
intrinsics = camera_trajectory_result["intrinsics"] # shape:  (3, 3, 3)
extrinsics = camera_trajectory_result["extrinsics"] # shape:  (3, 3, 4)
depth_meters_1 = depth_result["depth_meters"] # first image depth map
depth_meters_3 = depth_result_2["depth_meters"] # third image depth map

# Pixel coordinates (note:  depth_result is for first image, depth_result_2 for third)
# Pixel (1141, 320) in first image (depth_result)
# Pixel (1060, 47) in third image (depth_result_2)
# Note:  image dimensions are [height, width] = [966, 1302] per depth_result

# Convert pixel coordinates to 0-based indexing (if needed, but usually they are 0-based)
x1, y1 = 1141, 320 # first image
x3, y3 = 1060, 47 # third image

# Get depth at each pixel
depth_1 = depth_meters_1[y1, x1] # depth at pixel (1141, 320) in first image
depth_3 = depth_meters_3[y3, x3] # depth at pixel (1060, 47) in third image

# Compute local median depth for each point (for robustness, though not strictly needed for
distance)
depth_median_1 = np.median(depth_meters_1[y1-1:y1+2, x1-1:x1+2]) # 3x3 neighborhood
depth_median_3 = np.median(depth_meters_3[y3-1:y3+2, x3-1:x3+2]) # 3x3 neighborhood

# Print intrinsics (first image)
print("Intrinsics (first image):")
print(intrinsics[0])

# Print camera transforms (first and third images)
print("
Camera transforms (first image):")
print(extrinsics[0])
print("
Camera transforms (third image):")
print(extrinsics[2])

# Print depth local medians
print(f"
Depth local median at pixel (1141, 320) in first image:  {depth_median_1:.4f} meters")
print(f"Depth local median at pixel (1060, 47) in third image:  {depth_median_3:.4f} meters"
)

# Convert 2D pixel coordinates to 3D world coordinates using camera model
# For first image (image index 0)
K = intrinsics[0] # 3x3 intrinsic matrix
R1, t1 = extrinsics[0][:3, :3], extrinsics[0][:3, 3] # rotation and translation (world to
camera)

# Convert pixel to normalized image coordinates (assuming 0-based indexing)
u1, v1 = x1, y1
# Apply intrinsic matrix to get normalized coordinates
x_norm1 = (u1 - K[0, 2]) / K[0, 0] # (u - cx) / fx
y_norm1 = (v1 - K[1, 2]) / K[1, 1] # (v - cy) / fy
````
````

```
# Convert normalized image coordinates to 3D camera coordinates
# [Xc, Yc, Zc] = depth * [x_norm, y_norm, 1]
Xc1 = depth_median_1 * x_norm1
Yc1 = depth_median_1 * y_norm1
Zc1 = depth_median_1

# Convert camera coordinates to world coordinates
# Xw = R^T * (Xc - t)
Xw1 = np.dot(R1.T, np.array([Xc1, Yc1, Zc1]) - t1)
Xw1 = Xw1.flatten() # ensure 1D

# For third image (image index 2)
K = intrinsics[2] # 3x3 intrinsic matrix
R3, t3 = extrinsics[2][:3, :3], extrinsics[2][:3, 3] # rotation and translation (world to
camera)

# Convert pixel to normalized image coordinates
u3, v3 = x3, y3
x_norm3 = (u3 - K[0, 2]) / K[0, 0]
y_norm3 = (v3 - K[1, 2]) / K[1, 1]

# Convert normalized image coordinates to 3D camera coordinates
Xc3 = depth_median_3 * x_norm3
Yc3 = depth_median_3 * y_norm3
Zc3 = depth_median_3

# Convert camera coordinates to world coordinates
Xw3 = np.dot(R3.T, np.array([Xc3, Yc3, Zc3]) - t3)
Xw3 = Xw3.flatten() # ensure 1D

# Print derived 3D world coordinates
print(f"
3D world coordinates (first image point):  ({Xw1[0]:.4f}, {Xw1[1]:.4f}, {Xw1[2]:.4f})")
print(f"3D world coordinates (third image point):  ({Xw3[0]:.4f}, {Xw3[1]:.4f},
{Xw3[2]:.4f})")

# Compute Euclidean distance between the two 3D points
distance = np.linalg.norm(Xw1 - Xw3)
print(f"
Euclidean distance between the two points:  {distance:.4f} meters")
```

```
Execution output:
Intrinsics (first image):
[[1.2203016e+03 0.0000000e+00 6.5100000e+02]
 [0.0000000e+00 1.3876421e+03 4.8300000e+02]
 [0.0000000e+00 0.0000000e+00 1.0000000e+00]]

Camera transforms (first image):
[[ 0.8591257 0.2705432 -0.43440697 0.01150304]
 [-0.12202131 0.93264997 0.33952138 0.15892358]
 [ 0.49700487 -0.23868464 0.8342756 -0.07446227]]

Camera transforms (third image):
[[ 0.941648 0.22854781 -0.24711327 1.7913343 ]
 [-0.16772765 0.9551071 0.24420871 -0.2721306 ]
 [ 0.29183298 -0.18851092 0.93770844 1.2118572 ]]

Depth local median at pixel (1141, 320) in first image:  2.1547 meters
Depth local median at pixel (1060, 47) in third image:  4.0840 meters

3D world coordinates (first image point):  (1.8916, -0.6854, 1.3490)
3D world coordinates (third image point):  (0.5929, -1.6351, 2.5497)

Euclidean distance between the two points:  2.0076 meters
```

Example 15 · **Final answer**

WHAT HAPPENS The agent rounds the computed 2.0076 m to 2.0.

MODEL

```
<answer>2.0</answer>
```

What to notice. The two points are never visible in the same image, so the distance cannot be read off any one view. The harness supplies the three pieces a metric answer needs: camera poses relating the views (Step 1), metric depth in each view (Steps 2 and 3), and code that back-projects both points into a shared frame (Step 4). The model's own contribution is to locate the two markers in pixels and to state the computation. Tool results pass to the coding agent as named variables held in memory, such as `camera_trajectory_result` and `depth_result_2`, not as text the model must copy.

Trajectory: `outputs/gpt-6-sol/apiagentwithtools/..._sw13_100_tools/SparBench_tiny/full_trajectories/578.json`

END OF EXAMPLE 15

Example 16 Dataset: SparBench Model: GPT-6 Luna

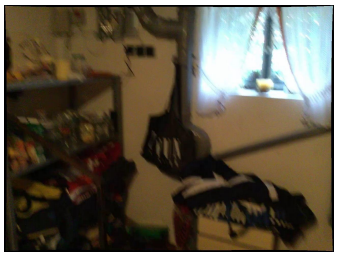
Overview (our summary)

Task. Three photographs of a basement storage area are taken from nearby positions; the first is the observer's view. A red point marks a cabinet, visible only in the third photo. Give the distance from the observer to the cabinet in metres.
Why it is hard. The cabinet is seen from a different position than the one the question asks about, so its depth in the third photo is not the answer; without tools the model answers 2.0 m.
How the agent solves it. It recovers the camera positions, reads the cabinet's depth in the third photo, and has the coding agent move that point into the observer's frame: 2.40 m, exactly the ground truth.

INPUT IMAGES



input_image_1. The observer's view.



input_image_2



input_image_3. The red point marks the cabinet, which appears only in this view.

QUESTION (as sent to the model)

```
Question:
From the person's position who took the photo, what is the distance to the center of
cabinet (red point)?  Express it in meters.  Calculate or judge based on the 3D center
points of these objects.  The first image becomes the primary perspective, correlating with
the observer's view.  Only one number should be included in your response.
Please answer the question using a single word or phrase.
```

WITHOUT TOOLS (GPT-6 Luna's full response to the same question)

```
2.0 m
```

RESULTS (MRA: SparBench's mean relative accuracy over thresholds 0.50–0.95)

	answer	MRA
GPT-6 Luna, no tools	2.0 m	0.00 ✗
GPT-6 Luna + OpenVLHarness	**2.4**	**1.00** ✓
Ground truth	2.4	

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
- input_image_2:  user_input
- input_image_3:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 16 · **Step 1** Camera Trajectory

WHAT HAPPENS Camera trajectory recovers where each photo was taken. The third was taken about 0.7 m behind and 0.5 m to the left of the observer.

ARGUMENTS

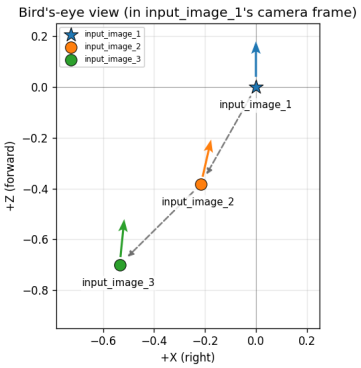
```
{
  "image_ids":  ["input_image_1", "input_image_2", "input_image_3"]
}
```

TOOL OUTPUT

```
Camera trajectory summary:
- input_image_1:  reference camera and reference view.
- input_image_2:
    camera movement:  salient direction:  none (all movements < 1 meter); detailed:  left
(-0.22), down (+0.12), backward (-0.38)
    camera rotation:  salient direction:  right (yaw +12.7°); detailed:  right (yaw
+12.7°), down (pitch +9.4°)
- input_image_3:
    camera movement:  salient direction:  none (all movements < 1 meter); detailed:  left
(-0.53), down (+0.11), backward (-0.70)
    camera rotation:  salient direction:  none (all rotations < 10°); detailed:  right
(yaw +5.7°), down (pitch +8.8°)
If no salient movement or rotation is reported, use the magnitude of each axis to determine
the most possible movement/rotation direction.

Bird's-eye view of the camera trajectory on input_image_1's X-Z plane is rendered in the
following image - refer as tool_generated_image_1 (dots = camera positions, forward-pointing
arrows = facing direction).

All vectors are relative to input_image_1's camera frame, using the coordinate system:  +X =
right, +Y = down, +Z = forward.  With this convention, positive yaw (rotation around Y
axis) = right and positive pitch (rotation around X axis) = down.  If the original question
assumes a different coordinate system (e.g.  with +Y = up and -Z = forward, which flips the
yaw/pitch signs), convert before answering.
```



Bird's-eye view of the three camera positions in input_image_1's frame.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_1, from Camera_
Trajectory_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- camera_trajectory_result:  dict (keys:  intrinsics, extrinsics, image_ids, num_images,
cameras, coordinate_system, reference_image_id)
```

Example 16 · **Step 2** Depth Estimation

WHAT HAPPENS Metric depth for the third photo, the only one showing the cabinet. The agent explains that it wants the depth at the red point.

ARGUMENTS

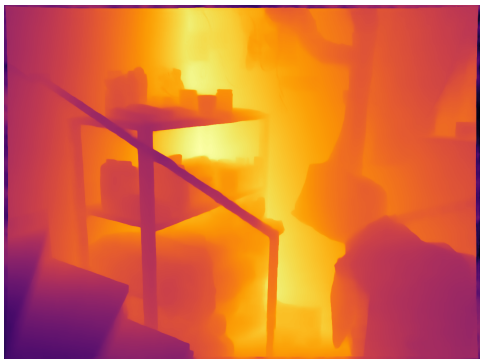
```
{
  "image_id":  "input_image_3",
  "colormap":  "inferno"
}
```

MODEL (text emitted with this call)

> *I'll estimate the marked cabinet's depth relative to the camera using a depth map of the image containing the red point.*

TOOL OUTPUT

```
Tool `Depth_Estimation_Tool` (tool_call_id=call_wl4V1aXn8L7m11iXLof958TR) returned:
Depth estimation complete.  Shape:  [966, 1302], colormap:  inferno.
Metric depth in meters available as `depth_meters` (float32 ndarray).
Colorized depth map rendered below; refer to it as tool_generated_image_2.
```



Colorized depth map of input_image_3.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_2, from Depth_
Estimation_Tool).
- depth_result:  dict (keys:  depth_shape, colormap, size, depth_meters)
```

Example 16 · **Step 3** **Python Coding Agent**

WHAT HAPPENS The coding agent is asked for the depth at the red point. The code fails because it uses a library it did not import.

ARGUMENTS

```
{
  "objective":  "Extract the estimated depth in meters at the red point marking the cabinet
in input_image_3 (roughly x=1110, y=923 in a 1302x966 image).  Report the local median depth
in a small surrounding patch, ignoring invalid values."
}
```

TOOL OUTPUT

```
Coding agent generated and executed the following Python code:
```

[The generated code, 28 lines, is not shown: this run failed with the error below.]

```
Execution error:  name 'math' is not defined
```

Example 16 · **Step 4** **Python Coding Agent**

WHAT HAPPENS A narrower retry reads the depth at the red point: 2.16 m. This is the distance from the third camera, not from the observer.

ARGUMENTS

```
{
  "objective":  "Read depth_result depth_meters at red point x≈1110 y≈923, image3 is
1302x966.  Compute median finite values in the nearby 7x7 patch.  Use no imports."
}
```

TOOL OUTPUT

```
Coding agent generated and executed the following Python code:
```python
# Get depth_meters array and dimensions
depth_map = depth_result["depth_meters"]
height, width = depth_result["depth_shape"]
```
```

```
# Target point (x≈1110, y≈923) - note:  image coordinates are (width, height) = (1302,
966)
x, y = 1110, 923

# Define 7x7 patch around the point (centered at (x, y))
patch_start_row = max(0, y - 3)
patch_end_row = min(height, y + 4)
patch_start_col = max(0, x - 3)
patch_end_col = min(width, x + 4)

# Collect finite values in the 7x7 patch
finite_values = []
for row in range(patch_start_row, patch_end_row):
    for col in range(patch_start_col, patch_end_col):
        val = depth_map[row, col]
        if not (val == 0 or val == float('inf') or val == float('-inf')):
            finite_values.append(val)

# Compute median of finite values
if len(finite_values) > 0:
    sorted_values = sorted(finite_values)
    n = len(sorted_values)
    median = sorted_values[n // 2] if n % 2 == 1 else (sorted_values[n // 2 - 1] + sorted_
values[n // 2]) / 2
else:
    median = None

print(median)
```


Execution output:
2.1564598
```

Example 16 · **Step 5** Python Coding Agent

**WHAT HAPPENS** The agent asks for the distance from the observer. The code turns the red point into a 3D point in the third camera's frame and moves it into the first camera's frame using the recovered camera positions: 2.3995 m.

**ARGUMENTS**

```
{
  "objective":  "Estimate distance from reference camera (image1) to cabinet red-point
center in image3.  Use its depth (local median around x=1110,y=923), intrinsics and
relative camera extrinsics/centers from camera_trajectory_result.  Unproject the pixel into
image3 camera coordinates, transform the 3D point into image1 camera coordinates, and
calculate Euclidean distance.  Report the computed value and relevant transform
conventions."
}
```

TOOL OUTPUT

```
Coding agent generated and executed the following Python code:
```python
import numpy as np

# Extract reference image index (image1) and target image index (image3)
ref_idx = 0 # 'input_image_1'
target_idx = 2 # 'input_image_3'

# Get depth at pixel (x=1110, y=923) in image3 (note:  depth is in meters)
depth_at_pixel = depth_result["depth_meters"][923, 1110] # y, x indexing

# Compute local median depth around the pixel for robustness (3x3 neighborhood)
y_center, x_center = 923, 1110
y_start, y_end = max(0, y_center - 1), min(depth_result["depth_shape"][0], y_center + 2)
x_start, x_end = max(0, x_center - 1), min(depth_result["depth_shape"][1], x_center + 2)
local_depth = depth_result["depth_meters"][y_start:y_end, x_start:x_end]
median_depth = np.median(local_depth)

# Use median depth for robustness
depth = median_depth

# Unproject pixel (x=1110, y=923) in image3 to 3D point in image3 camera coordinates
# Using intrinsics and depth
```

```
intrinsics = camera_trajectory_result["intrinsics"][target_idx] # 3x3
K = intrinsics # camera matrix

# pixel coordinates (0-indexed, so subtract 1 if needed, but depth_result uses 0-indexed)
x_pixel = x_center
y_pixel = y_center

# Unproject:  (X, Y, Z) = depth * (u - cx) / fx, depth * (v - cy) / fy, depth
fx, fy = K[0, 0], K[1, 1]
cx, cy = K[0, 2], K[1, 2]

X_3d = depth * (x_pixel - cx) / fx
Y_3d = depth * (y_pixel - cy) / fy
Z_3d = depth

point_3d_in_target_camera = np.array([X_3d, Y_3d, Z_3d])

# Transform point from image3 camera coordinates to image1 camera coordinates
# Use extrinsics:  world_to_camera = [R | t], so camera_to_world = R^T * (-t)
# We need to transform from image3 camera frame to image1 camera frame
# So:  point_in_ref_camera = R_ref_to_target @ point_in_target_camera + t_ref_to_target

# Get extrinsics for image3 (target) and image1 (reference)
extrinsics_target = camera_trajectory_result["extrinsics"][target_idx] # 3x4
extrinsics_ref = camera_trajectory_result["extrinsics"][ref_idx] # 3x4

# Extract rotation matrix and translation vector for target and reference
R_target = extrinsics_target[:, :3] # 3x3 rotation matrix
t_target = extrinsics_target[:, 3] # 3x1 translation vector

R_ref = extrinsics_ref[:, :3] # 3x3 rotation matrix
t_ref = extrinsics_ref[:, 3] # 3x1 translation vector

# Transform point from target camera frame to reference camera frame
# point_in_ref_camera = R_ref_to_target @ point_in_target_camera + t_ref_to_target
# R_ref_to_target = R_target^T (since extrinsics are world_to_camera)
# t_ref_to_target = -R_target^T @ t_target + t_ref

R_ref_to_target = R_target.T # rotation from target to ref
t_ref_to_target = -R_ref_to_target @ t_target + t_ref

# Apply transformation
point_3d_in_ref_camera = R_ref_to_target @ point_3d_in_target_camera + t_ref_to_target

# Compute Euclidean distance from reference camera center to the point
camera_center_ref = t_ref # camera center in reference frame is the translation vector
distance = np.linalg.norm(point_3d_in_ref_camera - camera_center_ref)

# Print results
print(f"Estimated distance from reference camera to cabinet red-point center in image3:
{distance:.4f} meters")
print(f"Transform conventions:  All vectors are relative to {camera_trajectory_
result['reference_image_id']}'s camera frame, using the coordinate system:  {camera_
trajectory_result['coordinate_system']}")
```

```
Execution output:
Estimated distance from reference camera to cabinet red-point center in image3:  2.3995
meters
Transform conventions:  All vectors are relative to input_image_1's camera frame, using the
coordinate system:  All vectors are relative to input_image_1's camera frame, using the
coordinate system:  +X = right, +Y = down, +Z = forward.  With this convention, positive
yaw (rotation around Y axis) = right and positive pitch (rotation around X axis) = down.
If the original question assumes a different coordinate system (e.g.  with +Y = up and -Z =
forward, which flips the yaw/pitch signs), convert before answering.
```

Example 16 · **Final answer**

WHAT HAPPENS The agent answers 2.4, which matches the ground truth.

MODEL

```
<answer>2.4</answer>
```

What to notice. The question asks for the distance from the observer of the first view, but the cabinet is marked only in the third view, taken about 0.7 m behind and 0.5 m to the left (Step 1). Depth read at the marker in view 3 is 2.16 m (Step 4), which is the distance from the wrong camera. The final code run unprojects the marker into view 3's camera frame and moves it into the observer's frame with the camera-trajectory extrinsics, giving 2.40 m (Step 5). A run that failed on a missing import (Step 3) is simply retried with a narrower objective.

Trajectory: `outputs/gpt-6-luna/apiagentwithtools/..._sw13_100_tools/SparBench_tiny/full_trajectories/605.json`

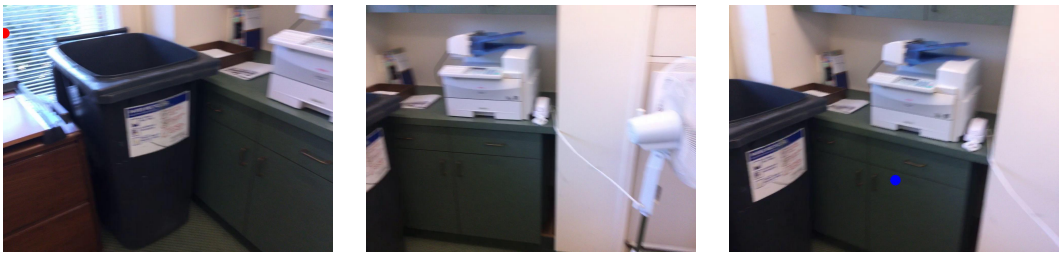
END OF EXAMPLE 16

Example 17 Dataset: SparBench Model: Qwen3-VL-32B

Overview (our summary)

Task. Three photographs of an office corner are taken from nearby positions. A red point marks a window in the first photo and a blue point marks a cabinet in the third. Give the 3D distance between them in metres. Without tools the same model answers 1.7 m.
How the agent approaches it. It chains four kinds of tool: camera trajectory to relate the views, depth in all three views, grounding to locate the window and the cabinet, and the coding agent to back-project both points into one frame and measure between them.
Caveat. The pipeline is the right one, but its last step does not use the marked points. The generated code reads each box as position plus width and height, although the boxes are corner coordinates, and it takes the first "cabinet" box, an upper wall cabinet at the top edge of the photo, instead of the lower cabinet with the blue point. The answer, 1.20 m against a ground truth of 1.1 m, is therefore close partly by coincidence.

INPUT IMAGES



input_image_1 input_image_2 input_image_3

QUESTION (as sent to the model)

```
Question:
In meters, how far is the center of window (red point) from the center of cabinet (blue
point)?  Calculate or judge based on the 3D center points of these objects.  Only one
numeric value is required for your response.
Please answer the question using a single word or phrase.
```

RESULTS (MRA: SparBench's mean relative accuracy over thresholds 0.50–0.95; no-tool answer from a separate run of the same model, final answer only)

	answer	MRA
Qwen3-VL-32B, no tools	1.7	0.00 ✗
Qwen3-VL-32B + OpenVLHarness	**1.1957**	**0.90** ✓
Ground truth	1.1	

With OpenVLHarness: full trajectory

HARNESS (environment update, shown in full)

```
Environment Update (not a new question):
Currently available image IDs for tool calls:
- input_image_1:  user_input
- input_image_2:  user_input
- input_image_3:  user_input
When a tool needs image input, pass one of these IDs via the *_image_id field.
Do NOT invent image file names or paths.
```

Example 17 · **Step 1** Camera Trajectory

MODEL (text emitted with this call)

> *<thinking>To calculate the distance between the center of the window (red point) and the center of the cabinet (blue point), I need to estimate their 3D positions. Since the images are from different viewpoints, I will first use the Camera_Trajectory_Tool to understand the camera motion between the frames. Then, I can use depth estimation and object grounding to get approximate 3D coordinates for both points. Finally, I'll compute the Euclidean distance between them.</thinking>*

ARGUMENTS

```
{
  "image_ids":  ["input_image_1", "input_image_2", "input_image_3"]
}
```

TOOL OUTPUT

```
Camera trajectory summary:
- input_image_1:  reference camera and reference view.
- input_image_2:
    camera movement:  salient direction:  none (all movements < 1 meter); detailed:  left
(-0.40), down (+0.07), backward (-0.29)
    camera rotation:  salient direction:  right (yaw +40.7°); detailed:  right (yaw
+40.7°), down (pitch +1.0°)
- input_image_3:
    camera movement:  salient direction:  none (all movements < 1 meter); detailed:  left
(-0.26), down (+0.03), backward (-0.15)
    camera rotation:  salient direction:  right (yaw +28.8°); detailed:  right (yaw
+28.8°), up (pitch -4.1°)
If no salient movement or rotation is reported, use the magnitude of each axis to determine
the most possible movement/rotation direction.

Bird's-eye view of the camera trajectory on input_image_1's X-Z plane is rendered in the
following image - refer as tool_generated_image_1 (dots = camera positions, forward-pointing
arrows = facing direction).

All vectors are relative to input_image_1's camera frame, using the coordinate system:  +X =
right, +Y = down, +Z = forward.  With this convention, positive yaw (rotation around Y
axis) = right and positive pitch (rotation around X axis) = down.  If the original question
assumes a different coordinate system (e.g.  with +Y = up and -Z = forward, which flips the
yaw/pitch signs), convert before answering.
```

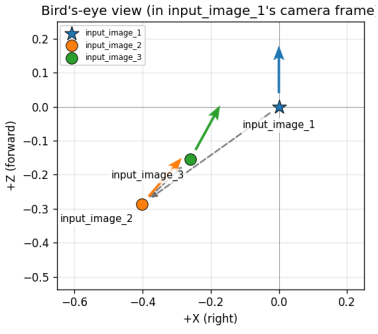


Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_1, from Camera_
Trajectory_Tool).
Available data bindings (auto-injected into Python_Coding_Agent_Tool):
- camera_trajectory_result:  dict (keys:  intrinsics, extrinsics, image_ids, num_images,
cameras, coordinate_system, reference_image_id)
```

Example 17 · **Step 2** Depth Estimation

MODEL (text emitted with this call)

> *<thinking>Now that I have the camera trajectory, I will estimate the 3D positions of the window (red point) and cabinet (blue point). I'll use depth estimation on each image to get depth values at these points, then combine with the camera positions and orientations to compute their 3D coordinates. Finally, I'll calculate the Euclidean distance between them.</thinking>*

ARGUMENTS

```
{
  "image_id": "input_image_1",
  "colormap": "inferno"
}
```

TOOL OUTPUT

```
Tool `Depth_Estimation_Tool` (tool_call_id=chatcmpl-tool-ba25e41bcbdd4895) returned:
Depth estimation complete. Shape: [966, 1302], colormap: inferno.
Metric depth in meters available as `depth_meters` (float32 ndarray).
Colorized depth map rendered below; refer to it as tool_generated_image_2.
```

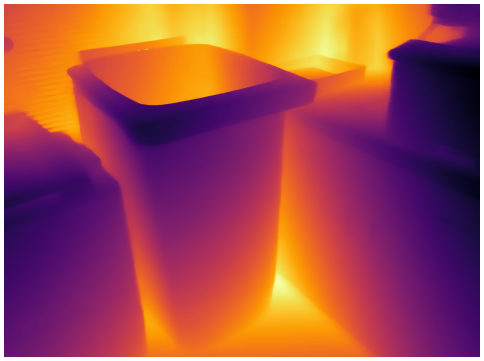


Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_2, from Depth_
Estimation_Tool).
- depth_result: dict (keys: depth_shape, colormap, size, depth_meters)
```

Example 17 · **Step 3** Depth Estimation

MODEL (text emitted with this call)

> *<thinking>Next, I will estimate the depth at the red point (window center) and blue point (cabinet center) in each image. Since the points are marked in different images, I'll need to use the depth maps from each respective image to get their depths. Then, using the camera positions and orientations from the trajectory, I can compute the 3D coordinates of each point and calculate the distance between them.</thinking>*

ARGUMENTS

```
{
  "image_id": "input_image_2",
  "colormap": "inferno"
}
```

TOOL OUTPUT

```
Tool `Depth_Estimation_Tool` (tool_call_id=chatcmpl-tool-a034e48c2e99033e) returned:
Depth estimation complete. Shape: [966, 1302], colormap: inferno.
Metric depth in meters available as `depth_meters` (float32 ndarray).
Colorized depth map rendered below; refer to it as tool_generated_image_3.
```

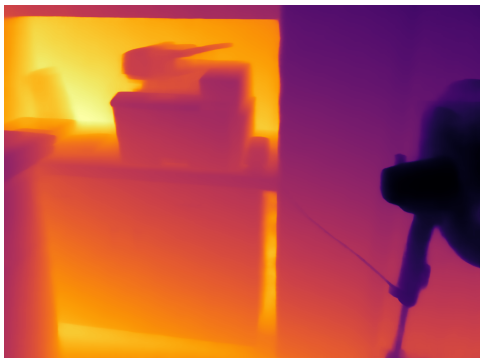


Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_3, from Depth_
Estimation_Tool).
- depth_result_2:  dict (keys:  depth_shape, colormap, size, depth_meters)
```

Example 17 · **Step 4** Depth Estimation

MODEL (text emitted with this call)

> *<thinking>Now, I will estimate the depth at the red point (window center) and blue point (cabinet center) in each image. Since the points are marked in different images, I'll need to use the depth maps from each respective image to get their depths. Then, using the camera positions and orientations from the trajectory, I can compute the 3D coordinates of each point and calculate the distance between them.</thinking>*

ARGUMENTS

```
{
  "image_id":  "input_image_3",
  "colormap":  "inferno"
}
```

TOOL OUTPUT

```
Tool `Depth_Estimation_Tool` (tool_call_id=chatcmpl-tool-956d2565d6a5d9fb) returned:
Depth estimation complete.  Shape:  [966, 1302], colormap:  inferno.
Metric depth in meters available as `depth_meters` (float32 ndarray).
Colorized depth map rendered below; refer to it as tool_generated_image_4.
```

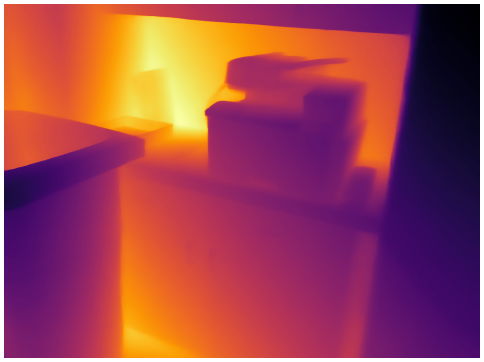


Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_4, from Depth_
Estimation_Tool).
- depth_result_3:  dict (keys:  depth_shape, colormap, size, depth_meters)
```

Example 17 · **Step 5** Visual Grounding

MODEL (text emitted with this call)

> *<thinking>Now I have depth maps for all three images. Next, I will use the Visual_Grounding_Tool to locate the window (red point) in input_image_1 and the cabinet (blue point) in input_image_3. This will give me the pixel coordinates of these points, which I can then use with the depth maps to get their 3D coordinates relative to each camera. After that, I'll use the camera trajectory data to transform these points into a common coordinate system and compute the distance between them.</thinking>*

ARGUMENTS

```
{
  "query": "window",
  "image_id": "input_image_1"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 1 objects for the prompt 'window'. Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: {'object 1': [0, 0, 353.92,
389.56]}. The object numbers correspond to the labeled masks rendered in the following
image, which you can refer to as tool_generated_image_5.
```

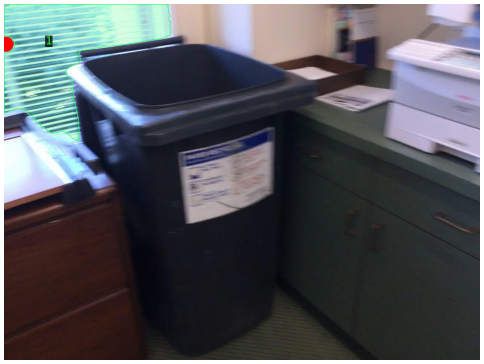


Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update: registers 1 new image in memory (tool_generated_image_5, from Visual_
Grounding_Tool).
- segment_window: dict (keys: pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 17 · **Step 6** Visual Grounding

MODEL (text emitted with this call)

> *<thinking>Now I will ground the cabinet (blue point) in input_image_3 to get its pixel coordinates. After that, I'll use the depth maps and camera trajectory data to compute the 3D coordinates of both points and calculate the distance between them.</thinking>*

ARGUMENTS

```
{
  "query": "cabinet",
  "image_id": "input_image_3"
}
```

TOOL OUTPUT

```
Visual_Grounding_Tool grounded 5 objects for the prompt 'cabinet'. Their corresponding
bounding boxes (on a 0-1000 scale, [x1, y1, x2, y2]) are: {'object 1': [194.95, 0, 373.1,
39.21], 'object 2': [252.05, 590.15, 416.32, 983.2], 'object 3': [382.59, 0, 593.34,
61.06], 'object 4': [423.93, 649.71, 721.75, 1000], 'object 5': [599.26, 0, 864.26,
```

```
83.16]}.  The object numbers correspond to the labeled masks rendered in the following
image, which you can refer to as tool_generated_image_6.
```

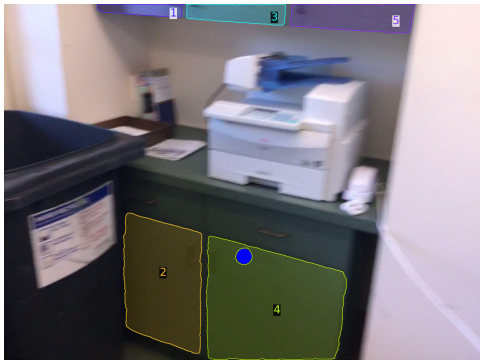


Image returned with this output.

HARNESS (environment update, summarized: new memory entries; new data bindings verbatim)

```
Memory update:  registers 1 new image in memory (tool_generated_image_6, from Visual_
Grounding_Tool).
- segment_cabinet:  dict (keys:  pred_boxes, pred_scores, pred_masks, num_masks, orig_img_h,
orig_img_w)
```

Example 17 · Step 7 Python Coding Agent

MODEL (text emitted with this call)

> *<thinking>Now I have the bounding boxes for both the window and cabinet. The red point (window center) is in input_image_1, and the blue point (cabinet center) is in input_image_3. I will use the depth maps for each image to get the depth at these points, then combine with the camera positions and orientations from the trajectory to compute their 3D coordinates. Finally, I'll calculate the Euclidean distance between them.</thinking>*

ARGUMENTS

```
{
  "objective":  "Compute the 3D coordinates of the window center (red point) in input_image_
1 and the cabinet center (blue point) in input_image_3 using the depth maps and camera
trajectory data.  Then, calculate the Euclidean distance between these two 3D points."
}
```

TOOL OUTPUT

```
Coding agent generated and executed the following Python code:
```

[The generated code, 126 lines, is not shown: this run failed with the error below.]

```
Execution error:  index 182417 is out of bounds for axis 0 with size 966
```

Example 17 · Step 8 Python Coding Agent

MODEL (text emitted with this call)

> *<thinking>The error occurred because the pixel coordinates for the window center exceeded the depth map dimensions. I need to adjust the pixel coordinates to fit within the depth map bounds. I'll use the original image dimensions to compute the center, then clamp the coordinates to the depth map size before accessing the depth value.</thinking>*

ARGUMENTS

```
{
  "objective":  "Compute the 3D coordinates of the window center (red point) in input_image_
1 and the cabinet center (blue point) in input_image_3 using the depth maps and camera
trajectory data.  Clamp the pixel coordinates to the depth map dimensions to avoid
out-of-bounds errors.  Then, calculate the Euclidean distance between these two 3D points."
}
```

TOOL OUTPUT

````
Coding agent generated and executed the following Python code:
```python
import numpy as np
from PIL import Image
import io
import base64

# Step 1:  Extract camera intrinsics and extrinsics for input_image_1 and input_image_3
# Note:  camera_trajectory_result["image_ids"] is ['input_image_1', 'input_image_2', 'input_
image_3']
# So index 0 is input_image_1, index 2 is input_image_3

K1 = camera_trajectory_result["intrinsics"][0] # 3x3 intrinsic matrix for input_image_1
K3 = camera_trajectory_result["intrinsics"][2] # 3x3 intrinsic matrix for input_image_3

# World-to-camera extrinsic matrices (OpenCV/COLMAP format:  [R | t])
# Note:  extrinsics are 3x4 matrices, where the first 3x3 is rotation and last column is
translation
# We need to convert to camera-to-world for 3D point reconstruction
# But for now, we'll use the extrinsics as is for projecting 3D points to camera space

# Extract extrinsics for input_image_1 and input_image_3
E1 = camera_trajectory_result["extrinsics"][0] # world-to-camera for input_image_1
E3 = camera_trajectory_result["extrinsics"][2] # world-to-camera for input_image_3

# Step 2:  Get window bounding box from segment_window (for input_image_1)
# The bounding box is in [x1, y1, w, h] format, normalized?  Let's check the values
# Preview:  [[-0.7693090438842773, -0.09972977638244629, 458.6855163574219,
377.0949401855469]]
# The x1, y1 are negative?  That suggests they are normalized or relative to center?
# But orig_img_h=968, orig_img_w=1296, and depth map is 966x1302 | very close.

# Let's assume the bounding box is in pixel coordinates, but the negative values suggest
it's not.
# Since the values are large (458, 377) and the image is 1296x968, it's likely absolute
pixel coordinates.
# But x1 is negative?  That's odd.  Let's check if it's normalized.

# Actually, looking at the values:  x1=-0.769, y1=-0.099 | these are very small, likely
normalized.
# But then w=458, h=377 | these are large, so probably not normalized.

# Let's assume the bounding box is in absolute pixel coordinates, and clamp to image size.
# We'll use the original image dimensions from segment_window for clamping.

orig_h = segment_window["orig_img_h"] # 968
orig_w = segment_window["orig_img_w"] # 1296

# Get window box
window_box = segment_window["pred_boxes"][0]
x1, y1, w, h = window_box

# Convert to center coordinates
window_center_x = x1 + w / 2
window_center_y = y1 + h / 2

# Clamp to image dimensions
window_center_x = np.clip(window_center_x, 0, orig_w - 1)
window_center_y = np.clip(window_center_y, 0, orig_h - 1)

# Step 3:  Get depth for window center in input_image_1
# depth_result is for input_image_1 (since it's the first depth result and we're using it
for input_image_1)
depth_map_1 = depth_result["depth_meters"]

# Clamp to depth map dimensions (966, 1302)
depth_h, depth_w = depth_map_1.shape
window_center_x_clamped = np.clip(window_center_x, 0, depth_w - 1)
window_center_y_clamped = np.clip(window_center_y, 0, depth_h - 1)

# Get depth at window center
depth_window = depth_map_1[int(window_center_y_clamped), int(window_center_x_clamped)]

# Step 4:  Convert 2D pixel + depth to 3D point in camera coordinate system (input_image_1)
# We need to use the intrinsic matrix K1 to unproject

# Convert pixel to normalized coordinates
u = window_center_x_clamped
v = window_center_y_clamped
````
````

```
# Unproject using depth and intrinsic matrix
# 3D point in camera coordinates:  [X, Y, Z] = depth * [ (u - cx) / fx, (v - cy) / fy, 1 ]
# But we need to invert the intrinsic matrix to get the 3D point

# Intrinsic matrix K1 is 3x3
fx, fy = K1[0, 0], K1[1, 1]
cx, cy = K1[0, 2], K1[1, 2]

# Compute 3D point in camera coordinates
X_cam = (u - cx) * depth_window / fx
Y_cam = (v - cy) * depth_window / fy
Z_cam = depth_window

# So 3D point in camera 1 coordinate system is [X_cam, Y_cam, Z_cam]
point_window_cam1 = np.array([X_cam, Y_cam, Z_cam])

# Step 5:  Get cabinet bounding box from segment_cabinet (for input_image_3)
# There are 5 detections, we'll use the first one (highest score)
cabinet_box = segment_cabinet["pred_boxes"][0]
x1_cab, y1_cab, w_cab, h_cab = cabinet_box

# Convert to center
cabinet_center_x = x1_cab + w_cab / 2
cabinet_center_y = y1_cab + h_cab / 2

# Clamp to image dimensions
cabinet_center_x = np.clip(cabinet_center_x, 0, orig_w - 1)
cabinet_center_y = np.clip(cabinet_center_y, 0, orig_h - 1)

# Step 6:  Get depth for cabinet center in input_image_3
depth_map_3 = depth_result_3["depth_meters"]

# Clamp to depth map dimensions
cabinet_center_x_clamped = np.clip(cabinet_center_x, 0, depth_w - 1)
cabinet_center_y_clamped = np.clip(cabinet_center_y, 0, depth_h - 1)

# Get depth at cabinet center
depth_cabinet = depth_map_3[int(cabinet_center_y_clamped), int(cabinet_center_x_clamped)]

# Step 7:  Convert 2D pixel + depth to 3D point in camera coordinate system (input_image_3)
# Use intrinsic matrix K3

fx3, fy3 = K3[0, 0], K3[1, 1]
cx3, cy3 = K3[0, 2], K3[1, 2]

# Unproject
X_cam3 = (cabinet_center_x_clamped - cx3) * depth_cabinet / fx3
Y_cam3 = (cabinet_center_y_clamped - cy3) * depth_cabinet / fy3
Z_cam3 = depth_cabinet

point_cabinet_cam3 = np.array([X_cam3, Y_cam3, Z_cam3])

# Step 8:  Convert both 3D points to world coordinates using extrinsics
# Extrinsics are world-to-camera:  E = [R | t], so to get world point from camera point:
# P_world = R^T * (P_cam - t) [since E * P_world = P_cam, so P_world = E^{-1} * P_cam]

# For input_image_1:  E1 = [R1 | t1], so E1^{-1} = [R1^T | -R1^T * t1]
R1 = E1[:, :3] # 3x3 rotation matrix
t1 = E1[:, 3] # 3x1 translation vector

# Invert E1:  E1_inv = [R1^T, -R1^T * t1]
R1_T = R1.T
t1_inv = -R1_T @ t1

# Convert point_window_cam1 to world coordinates
point_window_world = R1_T @ (point_window_cam1 - t1)

# For input_image_3:  E3 = [R3 | t3]
R3 = E3[:, :3]
t3 = E3[:, 3]

R3_T = R3.T
t3_inv = -R3_T @ t3

# Convert point_cabinet_cam3 to world coordinates
point_cabinet_world = R3_T @ (point_cabinet_cam3 - t3)

# Step 9:  Compute Euclidean distance between the two 3D points in world coordinates
distance = np.linalg.norm(point_window_world - point_cabinet_world)
```

```
# Output results
print(f"Window center 3D in world coordinates: {point_window_world}")
print(f"Cabinet center 3D in world coordinates: {point_cabinet_world}")
print(f"Euclidean distance between them: {distance:.4f} meters")

# Assign to variables for result namespace
window_3d_world = point_window_world
cabinet_3d_world = point_cabinet_world
distance_3d = distance
```

```
Execution output:
Window center 3D in world coordinates: [-0.59107007 -0.35877492 1.67238153]
Cabinet center 3D in world coordinates: [ 0.5192192 -0.77874663 1.52856489]
Euclidean distance between them: 1.1957 meters
```

Example 17 · **Final answer**

MODEL

> *<thinking>Based on the computation, the Euclidean distance between the center of the window (red point) and the center of the cabinet (blue point) is 1.1957 meters. This was calculated by estimating the 3D coordinates of both points using depth maps and camera trajectory data, then computing the distance in the world coordinate system.</thinking> <answer>1.1957</answer>*

What to notice. The tool chain is the one a metric cross-view distance needs: camera poses, depth in each view, grounding to find the objects, and code to combine them, and the model plans it in its own words before the first call. The weak link is the last step: the generated code misreads the box format and measures from the wrong cabinet, so the close agreement with the ground truth should not be read as a successful measurement.

Trajectory: `outputs/Qwen3-VL-32B-Instruct/baseagentwithtoolsandcode/..._20260526_095435_default/SparBench_tiny/full_trajectories/558.json`

END OF EXAMPLE 17

G BASELINE REPRODUCTION

We re-run every baseline ourselves on our benchmark suite using each method's official implementation (Table 22). All baselines receive exactly the same question text as OpenVLHarness, including each dataset's answer-format instruction (e.g., "a single integer", or a JSON list of boxes on a 0–1000 scale for grounding); the only addition is a short answer-format instruction that Codex appends for Qwen3-VL backbones (Appendix G.1). Their predictions are scored with the same evaluation pipeline and judges. Boxes that a baseline returns in a different coordinate frame, such as pixel coordinates of the model's resized input, are converted to our normalized format before scoring. Every baseline is evaluated on the full evaluation split. On Qwen3-VL, OctoTools and Visual Sketchpad follow the same three-run protocol as OpenVLHarness (Appendix H); runs with proprietary backbones and the training-based baselines are single runs. A request that fails or times out counts as incorrect.

Method	Backbone	Tools	Search
Training-free			
OctoTools	Qwen3-VL-8B/32B	Generalist solution generator	×
Visual Sketchpad	Qwen3-VL-8B/32B, GPT-6 Luna	Detection, SoM, depth, zoom, overlay	×
Codex	Qwen3-VL-8B/32B, GPT-6 Luna	Code execution (unavailable; see text)	×
Training-based (released checkpoints)			
AdaReasoner	AdaReasoner-7B	Point, OCR, crop, drawing	×
PixelReasoner	PixelReasoner-RL-v1 (7B)	Zoom-in, frame selection	×
DeepEyesV2	DeepEyesV2-7B	Code execution	×
OpenVLHarness	Matched to baseline	Nine tools (Table 15)	✓

Table 22: Configurations of the reproduced baselines. *Search* marks access to a web-search tool. All baselines except Codex decode greedily; Codex uses its own request settings.

G.1 Training-Free Harnesses

The training-free harnesses run on the same model endpoints as OpenVLHarness, so any difference reflects the harness rather than the model.

OctoTools. OctoTools (Lu et al., 2025) is a planner–executor framework in which a planner analyzes the query and proposes the next step, an executor turns it into a tool command, and a verifier decides whether to stop. We use the official implementation and route every language-model role, including the planner, executor, verifier, and the Generalist Solution Generator tool, to the Qwen3-VL backbone. We use its base toolset, the Generalist Solution Generator, with a budget of ten steps and greedy decoding. Because OctoTools accepts a single image per query, multi-image inputs are tiled into one composite with labeled panels ("Image 1", "Image 2", ...). The final answer is taken from its direct-output mode.

Visual Sketchpad. Visual Sketchpad (Hu et al., 2024) prompts the model to write Python that calls vision tools and draws on the image, in a ReAct-style loop. We use the official implementation with the Qwen3-VL backbone as its single planning agent and its vision-task toolset: open-vocabulary detection with Grounding DINO (Liu et al., 2024), sliding-window detection, Set-of-Mark segmentation (Yang et al., 2023a), depth estimation with Depth Anything (Yang et al., 2024), zoom-in, and image overlay, all served with the original expert models. The agent may take up to ten replies with greedy decoding, receives multiple images as separate inputs, and ends with an `ANSWER:` line from which we read the prediction. For grounding tasks, it is instructed to convert its detector's boxes to our output format.

Codex. Codex (OpenAI, 2025a) (v0.155.1) is OpenAI's general-purpose coding agent. We run it non-interactively (`codex exec`) in code mode, with a fresh, ephemeral session for every question, the question's images attached as inputs, and its final agent message taken as the prediction. Web search, browser use, apps, plugins, image generation, and sub-agents are disabled, and model-written code runs in a read-only sandbox. With GPT-6 Luna, Codex calls the OpenAI Responses API at medium reasoning effort and receives the question text unchanged. With Qwen3-VL-8B and Qwen3-VL-32B, it calls the same vLLM endpoints as OpenVLHarness through a small proxy that exposes Codex's code tool as a standard function tool, which our vLLM server otherwise fails to stream, and each response is capped at $2{,}048$ tokens. For the Qwen3-VL runs, Codex also appends a short instruction to the question: answer directly, give the option letter for multiple-choice questions, and use the image as evidence. A request that fails or exceeds the 300-second timeout counts as incorrect. LLM-judged benchmarks are scored with the Qwen3-VL-8B judge.

G.2 Training-Based Harnesses

For the training-based baselines, the tool-use behavior lives in the fine-tuned weights, so we evaluate each method's released checkpoint inside its authors' inference loop rather than re-training it. We followed a two-stage reproduction procedure. First, we executed each repository's own reproduction workflow—including its released environment, model checkpoint, prompt and image preprocessing, tool implementations, and agent loop—until the baseline ran correctly on its documented benchmarks. Second, we added a lightweight HTTP serving layer around that verified path.

The serving layer standardizes request transport, process lifecycle, and input normalization, while the model-specific reasoning logic remains in the released implementation. Each service accepts a question and one or more ordered images, returns a final prediction and an agent trajectory, and keeps the model and tool processes resident across examples.

All hosted baselines are evaluated through a shared external harness. For each benchmark example, the harness invokes the dataset's prompt builder, converts the resulting text and ordered images into a common request, and records the returned prediction. The harness controls outer concurrency and, after inference, writes the prediction file expected by the benchmark evaluator and invokes the dataset's native `evaluate` method using the same configured language-model judge where required. Crucially, scoring is performed entirely outside the baseline services: no service inspects ground truth, selects an evaluator, or computes accuracy. This separation ensures that all baselines are scored through a single consistent protocol. We compare them against OpenVLHarness on Qwen2.5-VL-7B-Instruct, the base model all three were fine-tuned from (Table 8).

AdaReasoner. AdaReasoner (Song et al., 2026) trains the model to adaptively compose tools. We use the AdaReasoner-7B checkpoint and the authors' `BaseToolInferencer`, which exposes pointing (backed by Molmo (Deitke et al., 2025)), OCR, cropping, and drawing and path-planning tools through `<tool_call>` blocks, for up to six rounds. AdaReasoner is the only baseline requiring two Conda environments: the model/vLLM stack and the GPU PaddleOCR tool workers have incompatible tested dependencies. The launcher starts a tool controller and dedicated Point, OCR, and Crop workers alongside the model server.

PixelReasoner. PixelReasoner (Wang et al., 2025a) is trained to reason in pixel space. We use the PixelReasoner-RL-v1 checkpoint and initialize the released Ray/vLLM engines, multimodal processor, and function-call prompt once at startup. The service retains the released tool parser and pixel-space operations (normalized cropping and frame selection). Inference proceeds at the granularity of model/tool rounds: ready requests are batched across vLLM engines, generated tool calls are executed, resulting crops or selected frames are appended to the conversation, and unfinished requests are requeued. The final answer is read from `\boxed{}`. The documented image-resolution budgets (`MIN_PIXELS`=401408, `MAX_PIXELS`=4014080) are preserved as they affect how much visual information reaches the model.

DeepEyesV2. DeepEyesV2 (Hong et al., 2026a) is trained for agentic multimodal reasoning with code execution. We use the released DeepEyesV2-7B checkpoint with the agent mode of the authors' vendored VLMEvalKit `VLLMAPIWrapper`. A single launcher supervises the text-search adapter, Python code service, one or more OpenAI-compatible vLLM model replicas, and the public inference API. The wrapper controls model calls, Python execution, retrieval turns, stopping rules, and answer extraction. All ordered input images are bound in the Python sandbox as `image_1`, `image_2`, ... so that generated code can address every input.

Fair-comparison controls. For a defensible comparison, the following are held fixed or explicitly reported across all baselines: dataset version, split, and prompt builder; model checkpoint and quantization; image preprocessing and resolution limits; context window, per-call generation cap, temperature, and maximum agent/tool rounds; enabled tools and external search backend; concurrency and failure-retry policy; judge model, judge prompt, and judge retry count; and whether failures are scored as incorrect or retried.

H Run-to-Run Variability

Agentic evaluation is stochastic: the backbone samples at a non-zero temperature, the harness makes further model calls inside its tools, and live search results change over time. This section describes how we account for this variability and quantifies it for the Qwen3-VL results in the main paper.

Protocol. Every Qwen3-VL number in the main paper, including the Base, OctoTools, Visual Sketchpad, and OpenVLHarness columns of Table 1 and every entry of Table 2, is the mean of three independent runs. Each run uses a different random seed, which controls both the backbone's sampling and every stochastic component of the harness, such as the model calls made inside the Webpage Visit and Python Coding Agent tools. All other factors are held fixed across runs: the

benchmark instances, prompts, tool backends, decoding parameters, iteration and context limits, and the scorers. Judges and answer extractors decode greedily, so the variation we measure comes from the agent rather than from scoring. We report all three runs of every configuration; no run is discarded or re-run, and a request that fails after retries is scored as incorrect in the run where it occurs. Search tools query the live web and cannot be seeded, so for search benchmarks the variation also reflects changes in retrieved content between runs. Results with proprietary backbones (GPT-5, GPT-6 Luna, and GPT-6 Sol) are single runs because of API cost.

Reported statistics. Tables 23 and 24 report, for each entry, the standard deviation of the score across the three runs. For averages over datasets, we first compute the average within each run and then take the standard deviation of these three averages; this is smaller than the per-dataset deviations because independent fluctuations partly cancel. With only three runs, each standard deviation is itself a coarse estimate and should be read as indicating the scale of run-to-run variation rather than an exact value.

		Qwen-3-VL-8B		Qwen-3-VL-32B	
Domain	**Dataset**	**Base**	**Ours**	**Base**	**Ours**
Counting and Grounding	FSC-147	0.26	0.17	0.26	0.59
	CountQA	0.40	1.08	0.40	0.53
	PixmoCount	1.64	1.53	1.64	0.22
	OdinW-13	0.98	0.94	0.89	1.94
	FSCD-147	0.14	0.06	0.14	0.10
	FSCD-LVIS	0.15	0.14	0.15	0.22
Visual Search and Deep Research	MMSearch	0.00	4.94	0.00	1.08
	MMSearch-Plus	0.40	1.27	0.25	0.61
	RealX-Bench	1.10	1.60	1.10	2.01
	FVQA	1.41	3.93	1.41	0.67
	HLE-VL	0.00	0.87	0.00	1.37
	BC-VL	1.41	0.77	1.41	0.29
General VQA and Hallucination	RealWorldQA	0.80	2.27	0.80	0.57
	SimpleVQA	0.07	0.69	0.07	0.60
	HallusionBench	0.64	0.61	0.64	1.64
	FREAK-VQA	0.67	0.65	0.67	0.80
	FREAK-MCQ	0.70	0.61	0.70	0.35
	V*	1.11	0.91	1.11	2.12
Spatial Understanding	MMSI-Bench	0.40	0.62	0.40	0.31
	ERQA	0.18	1.95	0.18	2.29
	EmbSpatial	0.10	0.34	0.10	0.22
	SparBench	0.36	1.47	0.35	1.67
	MindCube	0.59	1.08	0.59	1.15
	Average	0.16	0.35	0.16	0.24

Table 23: Standard deviations across three runs for the Qwen3-VL Base and OpenVLHarness results in Table 1. *Average* is the standard deviation of the per-run 23-dataset average.

Main results. Per-dataset standard deviations are small relative to the differences between methods (Table 23). For OpenVLHarness, most are below 2 points; the largest occur on search benchmarks with live retrieval, such as MMSearch (171 questions, where one question is worth 0.58 points) and FVQA with Qwen3-VL-8B. OpenVLHarness varies somewhat more than Base, as expected for multi-turn trajectories in which an early difference in a tool call can change the rest of the trajectory. The 23-dataset averages are very stable, with standard deviations of at most 0.35 points. The gain of OpenVLHarness over Base in this average ($+11.2$ at 8B and $+12.4$ at 32B) is therefore about 29 and 43 times the combined standard deviation of the two averages, so the improvement cannot be attributed to run-to-run variation.

Variant	Counting and Grounding		Visual Search and Deep Research		General VQA and Hallucination		Spatial Understanding		Overall	
	8B	32B	8B	32B	8B	32B	8B	32B	8B	32B
Baseline (no tool)	0.64	0.63	0.46	0.46	0.36	0.36	0.24	0.24	0.22	0.22
+ Tool	0.23	0.19	1.11	0.75	0.40	0.43	0.59	0.73	0.34	0.29
+ Capability layer	0.27	0.31	0.71	0.64	0.36	0.42	0.55	1.01	0.25	0.32
+ Textual memory	0.22	0.20	0.74	0.56	0.56	0.46	0.31	0.39	0.25	0.21
OpenVLHarness	0.27	0.24	1.05	0.65	0.62	0.44	0.56	0.75	0.34	0.28

Table 24: Standard deviations across three runs for the ablation in Table 2. Domain and overall columns are standard deviations of the per-run averages.

Ablation. The overall averages in Table 2 have standard deviations between 0.2 and 0.35 points for every variant and backbone (Table 24). Adding tools, the capability layer, and multimodal memory each changes the overall average by several points, far beyond this variation. The textual-memory step ($+0.7$ at 8B and $+0.3$ at 32B) is comparable to run-to-run variation, which is consistent with our conclusion that text descriptions of images are a weak substitute for retaining the images themselves.

Single-run results. Proprietary backbones are evaluated once, so their per-dataset numbers carry run-to-run variation of the scale measured above. Their conclusions, however, rest on large differences: OpenVLHarness improves the 23-dataset average by $+12.6$ points with GPT-6 Luna and $+11.2$ with GPT-6 Sol, more than thirty times the largest average-level standard deviation we observe with Qwen3-VL.

I Additional Related Work

Multimodal search agents. Our work targets multimodal search, where a query is resolved by orchestrating reasoning, perception, and live web retrieval across many turns. A growing collection of benchmarks frames this task: MMSearch (Jiang et al., 2024) couples a multimodal query with a curated search engine; MMSearch-Plus (Tao et al., 2026) adds provenance-aware verification; BrowseComp-VL (Geng et al., 2026) and MM-BrowseComp (Li et al., 2025b) raise cross-modal information-seeking difficulty; VisBrowse-Bench (Zhang et al., 2026c) targets visual-native search; and RealX-Bench (Hong et al., 2026a) couples fine-grained perception with multi-hop retrieval. Beyond pure search, the landscape extends to embodied tasks that integrate web information with physical action (Hong et al., 2025). Recent browsing agents tackle these benchmarks via hand-engineered tool inventories (Geng et al., 2026; Huang et al., 2026; Yao et al., 2026; Chen et al., 2026; Wu et al., 2025a; Narayan et al., 2025), supervised fine-tuning or reinforcement learning that teaches tool use, or memory learned over a fixed inventory (Xiao et al., 2025; Hasegawa et al., 2025). Our contribution is orthogonal: rather than freeze the inventory, we *generate* it.

Counting and grounding. Class-agnostic counting was popularized by FSC-147 (Ranjan et al., 2021), which asks for the number of objects of an arbitrary category; specialized counters such as CounTR (Liu et al., 2022), LOCA (Djukic et al., 2023), and CountGD (Amini-Naieni et al., 2024) regress density maps or detections, and FSCD (Nguyen et al., 2022) extends the task to predicting a box for every instance. Open-vocabulary detectors and segmenters, including GLIP (Li et al., 2022), Grounding DINO (Liu et al., 2024), and SAM 3 (Carion et al., 2025), localize arbitrary text-specified categories, and MLLMs have begun to count by pointing (Deitke et al., 2025). Nevertheless, MLLMs remain unreliable once scenes contain more than a handful of objects (Paiss et al., 2023; Tamarapalli et al., 2025). Tool-use systems such as ViperGPT (Surís et al., 2023), VisProg (Gupta & Kembhavi, 2023), and Visual Sketchpad (Hu et al., 2024) let a language model call detectors and count their outputs programmatically. OpenVLHarness follows this direction but keeps grounding outputs as persistent evidence: boxes, masks, and scores are stored as typed bindings that the Python Coding Agent can filter, deduplicate, and aggregate across turns, instead of a count that is read once from text.

Spatial Reasoning. Spatial reasoning benchmarks test models beyond static single-image relations. EmbSpatial-Bench (Du et al., 2024) and ERQA (Team et al., 2025) target spatial and embodied reasoning in simulated and real-world scenes; SPAR-Bench (Zhang et al., 2026a) covers diverse single- and multi-view spatial tasks; and MindCube (Wang et al., 2025c) and MMSI-Bench (Yang

et al., 2025b) emphasize reasoning across multiple or limited views. In parallel, VGGT (Wang et al., 2025b) and subsequent VGGT-style geometric foundation models (Hu et al., 2026) recover explicit scene geometry, including depth, camera pose, point maps, and tracks from visual observations. OpenVLHarness treats such geometric models as callable capabilities rather than requiring the MLLM itself to internalize 3D reconstruction, allowing their outputs to be composed with grounding and other persistent evidence during reasoning.

Experience Learning and Harness Optimization. Learning from past experience has a long history in reinforcement learning, exemplified by Hindsight Experience Replay (Andrychowicz et al., 2017), which relabels failed rollouts as successes, STaR (Zelikman et al., 2022), which bootstraps a reasoner on its own correct rationales, and ARMAP (Chen et al., 2025b), which learns a reward model from an agent's own trajectories to guide its planning. A newer line treats memory as a learning signal, distilling reusable knowledge from cross-sample working memory, though these methods often require additional training or model-weight updates (Packer et al., 2023; Yan et al., 2025b; Zhou et al., 2025b; Wang et al., 2024b; Yang et al., 2025a; Ouyang et al., 2026; Zhang et al., 2025; Yan et al., 2025a; Ren et al., 2026; Shi et al., 2026; Hong et al., 2026b; Jiang et al., 2026). A third line instead optimizes prompts and harnesses without any model-weight updates, using LLM- and coding-agent-driven optimizers that rewrite prompts, pipelines, and harness code from observed failures (Khattab et al., 2024; Yuksekgonul et al., 2024; Agrawal et al., 2025; Lee et al., 2026; Yuksekgonul et al., 2026; Sharma, 2025), but to date this is done purely in the text-only prompt and harness domain.

J LIMITATIONS

Bounded by the underlying tools. OpenVLHarness improves how evidence from expert models is gathered, stored, and composed, but it cannot exceed what those experts perceive. Counting and grounding at very high object counts remain largely unsolved (Appendix B.4), and on several perception-bound VQA and spatial datasets generic tool use adds little or slightly hurts (Section 3).

Cost and latency. Multi-turn tool use, search, and code execution make each query considerably more expensive and slower than direct inference, and the expert models add GPU serving requirements. We cap trajectories at fifteen iterations; longer trajectories would also need a memory eviction or summarization policy, which we leave to future work.

Reproducibility of live search. Search benchmarks depend on commercial search APIs and live web content, which change over time, so exact numbers can drift between runs. We mitigate this by averaging three runs for all Qwen3-VL results (Appendix H), but results obtained at a later date may differ.

Evaluation. Open-ended answers are scored by LLM judges, which can occasionally misjudge paraphrased or partially correct answers, and proprietary models sometimes refuse identity-related questions (Appendix D.2). Baselines are re-run with their official code on our benchmark suite and prompts, which may differ from the settings their authors tuned for.