ChartAgent: A Multimodal Agent for Visually Grounded Reasoning in Complex Chart Question Answering

Anonymous ACL submission

Abstract

Recent multimodal LLMs have shown promise in chart-based visual question answering, but their performance declines sharply on unannotated charts—those requiring precise visual interpretation rather than relying on textual shortcuts. To address this, we introduce **ChartAgent**, a novel agentic framework that explicitly performs visual reasoning directly within the chart's spatial domain. Unlike textual chain-of-thought reasoning, ChartAgent iteratively decomposes queries into visual subtasks and actively manipulates and interacts with chart images through specialized actions such as drawing annotations, cropping regions (e.g., segmenting pie slices, isolating bars), and localizing axes, using a library of chart-specific vision tools to fulfill each subtask. This iterative reasoning process closely mirrors human cognitive strategies for chart comprehension. ChartAgent achieves state-of-the-art accuracy on the ChartBench and ChartX benchmarks, surpassing prior methods by up to 16.07% absolute gain overall and 17.31% on unannotated, numerically intensive queries. Our work is among the first to demonstrate visually grounded reasoning for chart understanding using tool-augmented multimodal agents. All resources (code, data, agent trajectories, and demos) available at: https://anonymous.4open.science/r/ChartAgent.

1 Introduction

Charts, including bar plots, pie charts, line graphs, and their many variants, are foundational tools for communicating quantitative information across domains such as finance, science, and journalism (Chishtie et al., 2022; Srivastava et al., 2025). Enabling computational systems to answer natural-language questions about charts, referred to as *chart visual question answering* (Chart VQA), remains an essential yet challenging problem in multimodal machine learning research (Masry et al., 2022; Xu et al., 2023; Xia et al., 2024; Wang et al., 2024b). Recent advances in multimodal large language models (MLLMs) have driven substantial progress in general visual reasoning tasks (Liu et al., 2023d; Hurst et al., 2024; Li et al., 2024). However, their performance degrades significantly on Chart VQA, especially when dealing with charts that lack explicit textual annotations of key values or labels, commonly referred to as *unannotated* charts (Xu et al., 2023; Xia et al., 2024; Islam et al., 2024) (see Appendix A for examples). These scenarios demand accurate visual grounding and interpretation (e.g., estimating numerical values from graphical elements), a setting where even state-of-the-art (SoTA) MLLMs often struggle.

To address these shortcomings, we draw inspiration from how humans reason with charts. Humans typically process graphical elements sequentially, interpreting axes, legends, and segments, and often add annotations to support intermediate reasoning, such as tracing bars and lines to compare values, circling or shading pie slices to judge proportions, and highlighting legends or markers to align categories. Building on these cognitive strategies, we propose **ChartAgent**, a novel agentic framework explicitly designed for visually grounded reasoning in the chart domain (see Figure 1). At the core of ChartAgent lies a multi-turn interaction loop that progressively decomposes chart queries into subtasks that are primarily visual and occasionally numerical, while simultaneously manipulating and interacting with chart images through precise, modular perception tools tailored to fulfill these subtasks, thereby augmenting MLLM reasoning with chart-specialized visual capabilities. To the best of our knowledge, and complementary to existing chart VQA approaches that rely on prompting or fine-tuning MLLMs (Masry et al., 2025, 2024; Han et al., 2023; Liu et al., 2023b), this work is the first to demonstrate visually grounded reasoning for chart understanding through tool-augmented

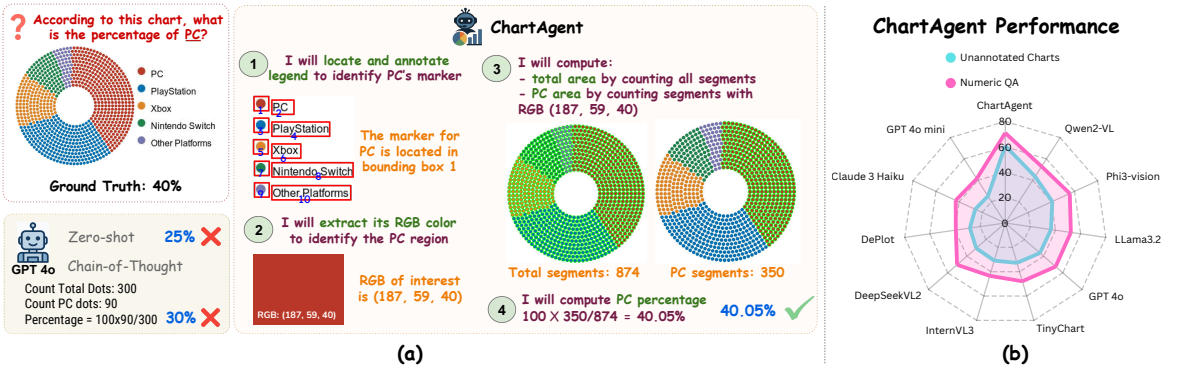


Figure 1: **Comparison of our work with the existing SoTA. (a)** `ChartAgent` performs visually grounded reasoning in the chart domain. For this unannotated chart, GPT-4o fails to produce the correct answer, whereas `ChartAgent` succeeds. **(b)** `ChartAgent` performance on unannotated charts and numeric QA compared with the top-10 SoTA.

multimodal agents, achieving SoTA performance. Overall, our **key contributions** are:

- ***Multimodal Agent for Charts:*** We introduce `ChartAgent`, the first framework to augment MLLM reasoning with chart-specialized visual capabilities for Chart VQA, systematically demonstrating visually grounded reasoning in charts via a tool-augmented multimodal agent.
- ***Modular Vision Tool Library with Self-Verification:*** An agent-compatible library of chart-specialized perception tools covering 40+ chart types, generating interpretable visualizations (see Figures 8, 9) that not only support grounded reasoning in `ChartAgent` but also enable a visual self-verification mechanism, allowing the agent to inspect intermediate results and adaptively adjust reasoning and tool use.
- ***State-of-the-Art Performance:*** `ChartAgent` achieves new SoTA, surpassing 30+ baselines by up to 16.07% absolute gain overall and 17.31% on unannotated, numerically intensive queries, evaluated on the well-established ChartBench and ChartX datasets spanning 40+ chart types.
- ***In-Depth Analysis:*** We conduct extensive analyses to demonstrate the effectiveness of `ChartAgent`. Specifically, we show that (a) it is effective across diverse chart types, (b) it achieves the highest scores across varying visual and reasoning complexity levels of chart–QA pairs, and (c) it serves as a plug-and-play framework that enhances performance across different base MLLMs, thereby validating both effectiveness and generalization. We also present a failure mode analysis highlighting common errors.

2 Related Work

Chart VQA Chart VQA interprets charts to answer natural-language queries. Early synthetic datasets (Kahou et al., 2017; Kafle et al., 2018) emphasized visual reasoning but lacked real-world diversity. Later benchmarks (Methani et al., 2020; Masry et al., 2022; Huang et al., 2024; Xu et al., 2023; Xia et al., 2024; Wang et al., 2024b) introduced realistic, diverse, and numerically intensive charts. Chart-specific MLLMs (Zhang et al., 2024; Masry et al., 2023; Liu et al., 2024a; Masry et al., 2024) enhanced instruction tuning and vision–language alignment, while hybrid approaches (Luo et al., 2021) integrated vision tools with rule-based parsing. However, recent studies (Xu et al., 2023; Razeghi et al., 2024; Islam et al., 2024) reveal sharp performance drops on unannotated charts, highlighting poor visual grounding. Our work addresses this gap through chart-specialized, visually grounded reasoning.

MLLMs and Visual Grounding General-purpose MLLMs such as GPT-4 (Achiam et al., 2023), GPT-4o (Hurst et al., 2024), Gemini (Team et al., 2023), LLaVA (Liu et al., 2023d), and Visual CoT (Shao et al., 2024) have advanced visual reasoning. For stronger grounding, models integrate tools or visual prompts: Visual ChatGPT (Wu et al., 2023), MM-ReAct (Yang et al., 2023b), ViperGPT (Surís et al., 2023), and VisProg (Gupta and Kembhavi, 2023) employ structured tools, while Visual Sketchpad (Hu et al., 2024b) and Set-of-Marks (Yang et al., 2023a) iteratively refine and annotate inputs. Inspired by these, our approach unites iterative reasoning, visual prompting, and modular vision tools for chart-grounded understanding.

Agentic Frameworks Agent-based AI systems, defined by perception, cognition, and action, have advanced with LLM integration. The ReAct framework (Yao et al., 2023) structures interactions into iterative reasoning, action, and observation, while platforms such as AutoGen (Wu et al., 2024a),

CrewAI (cre), LangChain (Lan), LangGraph (lan), and AutoGPT (aut) support practical implementations. MLLM agents extend this paradigm to robotics (Nasiriany et al., 2024; Hori et al., 2025), vision-language reasoning (Liu et al., 2025; Yang et al., 2023b), and GUI navigation (Verma et al., 2024; He et al., 2024). Similarly, ChartAgent integrates multimodal reasoning with modular, chart-oriented vision tools in an agentic framework. See Appendix B for an extended review.

3 ChartAgent: A Multimodal Agent for Visually Grounded Reasoning in Charts

Given a multimodal query consisting of a chart image and a natural-language question about the chart, the goal is to generate an answer that accurately reflects the information conveyed in the chart. Building on human strategies for chart comprehension, such as highlighting legend entries to clarify category mappings, sketching guide lines across bars or axes to compare values, or shading portions of a pie chart to approximate proportions, we propose **ChartAgent**. As illustrated in Figure 2, ChartAgent is a novel agentic framework that equips MLLMs with structured visual reasoning capabilities for charts, by decomposing queries into visual subtasks and directly interacting with chart images in their spatial domain through specialized vision tools to accomplish these subtasks. These tools are supported by interpretable intermediate visualizations that enable adaptive refinement of reasoning and grounding until a confident answer is reached or the iteration limit is exhausted.

Visually Grounded Chart Reasoning The foundation of ChartAgent is a structured, iterative ReAct (Yao et al., 2023)-style multi-turn interaction loop within the chart's visual environment, which at each time step t generates a sequence of Thought, Action, and Observation phases to guide the agent in interpreting charts and answering user queries.

- **Thought (Reasoning)**: The MLLM evaluates the current state s_t, which includes the multimodal query along with previous thoughts, actions, and observations, to derive the next subtask (goal) g_t that guides the subsequent action toward answering the user's query. These sub-goals primarily involve visual perception tasks (e.g., segmenting chart elements, detecting and annotating legends, or localizing axes), but may also include numerical operations (e.g., interpolation, arithmetic).
- **Action (Chart Tool Execution)**: Based on the subtask g_t from the Thought phase, the agent selects and executes an appropriate tool $a_t^{\text{chart-tool}}$ from a modular chart-specialized library (see App. Table 5) that directly manipulates the chart image. Examples include pie segmentation, bar isolation, legend detection, axis tick localization, and interpolation. Each tool returns structured outputs (e.g., numeric estimates, labels, detected coordinates) and, when applicable, interpretable intermediate or final visualizations (e.g., segmentation masks with labels, colored overlays for pie slices, bar height markers, annotated legends, or bounding boxes) (see App. Figures 8, 9), which support the agent's subsequent visual self-verification.
- **Observation (Visual Self-Verification and Adaptive Tool Use)**: Based on the invoked action $a_t^{\text{chart-tool}}$, ChartAgent receives new perception-friendly visualizations and outputs o_{t+1}. The multimodal state is then updated as $s_{t+1} = (s_t, g_t, a_t^{\text{chart-tool}}, o_{t+1})$. ChartAgent then interprets and verifies these multimodal outputs, particularly for perception-related tools, by visually inspecting the provided visualizations to assess their accuracy. If verification reveals unsatisfactory results (e.g., incomplete segmentation, mismatched legend associations, overly small pie slices, incorrect colors, negative bar heights, or outputs inconsistent with axis values), the agent adaptively adjusts its tool use in the next iteration $t+1$, for instance by invoking an alternative tool or tweaking parameters such as detection thresholds. This iterative correction loop mimics human-like debugging, enabling ChartAgent to reason and ground with visualizations it generated on the chart, thereby ensuring improved chart VQA capabilities (see Sec. 5.3).

Chart Interaction and Manipulation The effectiveness of ChartAgent hinges on the careful design of a modular library of perception and numeric tools tailored for chart understanding (a detailed taxonomy is provided in App. F and Table 5). Inspired by primitive visual tasks in natural image domains (e.g., object detection, segmentation, relational inference), we define analogous primitive tasks for the chart domain, treating chart elements (e.g., bars, pie slices, lines, legends, tick marks, and axis labels) as fundamental visual "objects." By targeting shared components such as legends, axes, ticks, bar segments, and pie slices, these tools enable broad generalization across diverse chart formats (see App. D for the 40+ chart types supported in ChartAgent). Guided by this perspective,

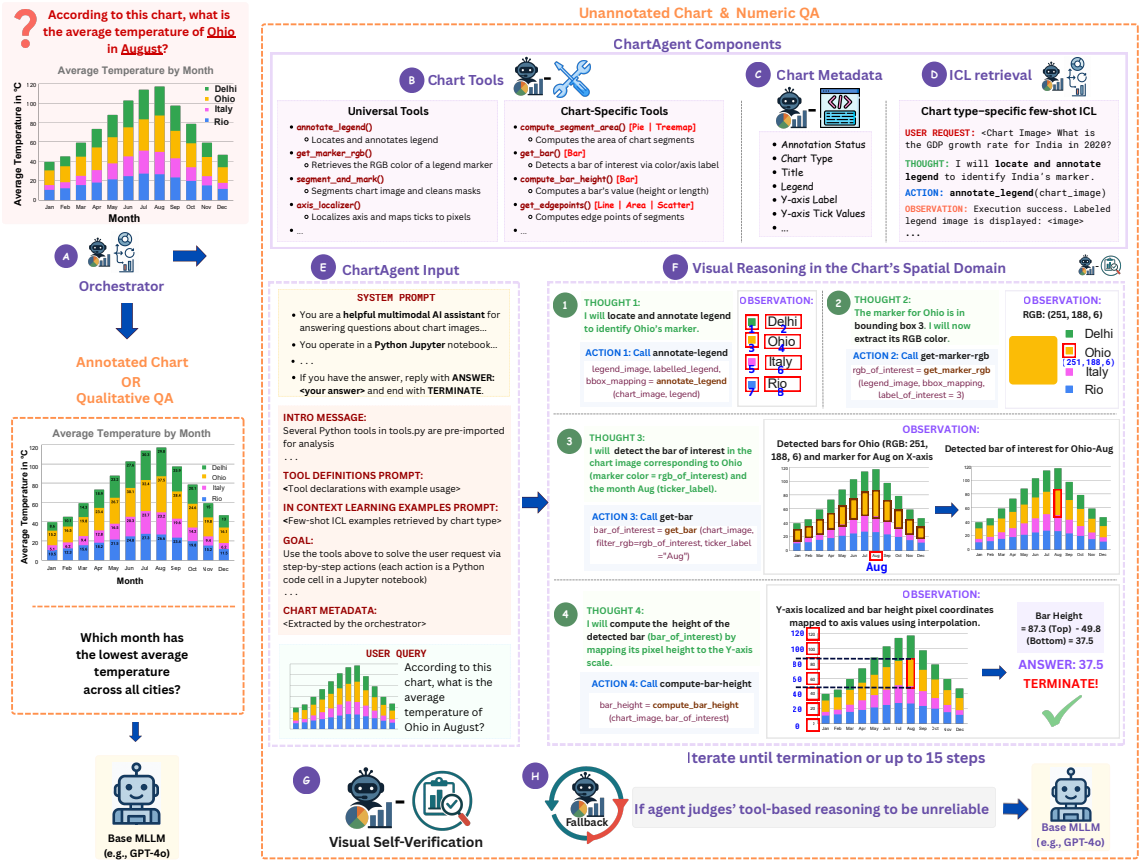


Figure 2: **ChartAgent.** The **(A)** orchestrator extracts chart metadata and routes annotated charts with textual shortcuts and qualitative QA to the base MLLM, while unannotated charts and numeric queries trigger the ReAct-style loop. The system includes **(B)** a library of universal and chart-specific tools, **(C)** metadata for parameterizing tool usage and retrieving chart-type-specific ICL examples, and **(D)** few-shot ICL retrieval. Using these components as the **(E)** input, ChartAgent performs **(F)** iterative visual reasoning, supported by **(G)** visual self-verification of intermediate outputs. When tool-based reasoning is unreliable, **(H)** the agent falls back to the base MLLM.

we designed a structured, chart-domain-specific set of *primitive* tools, organized into two categories:

1. **Universal chart tools:** General-purpose perception tools applicable across chart types, such as segmentation, legend detection, axis localization, and numeric interpolation.
2. **Chart-specific tools:** Tools specialized for particular chart types (e.g., pie, bar, line, box), targeting subtasks unique to their visual structures.

Each tool is deliberately scoped to remain clear and distinct, avoiding overly fine-grained or excessively complex functionalities, thereby ensuring robust implementations with modern vision techniques.

Architecture and Components

- **Chart Metadata Extraction and Orchestration**: ChartAgent begins with an LLM-based orchestrator (e.g., GPT-4o) that extracts comprehensive chart metadata, including chart type, title, legend details, axis labels and tick marks, annotation status (annotated or unannotated), and a concise visual description (see App. M.1.3). This metadata is critical for orchestrating the smart routing mechanism, which first determines whether perception tools are necessary for the user task. For annotated charts containing explicit textual shortcuts (e.g., numerical annotations or clear labels) or for queries requiring mainly qualitative reasoning, direct reasoning by the base MLLM is often sufficient. In such cases, the orchestrator routes the query directly to the MLLM balancing accuracy and computational efficiency. In contrast, for unannotated charts (see App. A), where accurate interpretation of graphical elements, such as bar-height/pie-area estimation, or legend association, is essential, the orchestrator initiates a deeper, iterative routine of visual reasoning to derive the answer. In the unannotated case, the extracted metadata is also used to retrieve appropriate chart-type-specific few-shot in-context learning (ICL) examples and to parameterize subsequent tool usage.
- **Chart Tools Implementation**: Chart tools are implemented as Python functions callable by ChartAgent. Some of these tools internally leverage SoTA computer vision and OCR methods, such as Segment Anything (SAM) (Kirillov et al., 2023),

Semantic SAM (Li et al.), Tesseract (tes), and EasyOCR (eas). They also handle edge cases (e.g., rotated text, fuzzy label matching for legends or axis ticks, and filtering small, background, or overlapping segments) and return structured outputs (e.g., numeric values, bounding boxes, text labels) along with visualizations (e.g., segmentation masks with labels or bounding box annotations; see details and examples in App. F.2 and Figures 8, 9) that are explicitly designed to facilitate `ChartAgent`'s visual self-verification. See App. F for detailed tool descriptions, App. M.1.2 for prompt details, and Sec. 5.3 for an analysis of their effectiveness.

- **ICL**: `ChartAgent` uses few-shot (1–2) ICL examples that are specifically retrieved based on the chart type identified during metadata extraction (see App. M.1.4). If no ICL examples exist for the detected chart type, then none are added.
- **Multimodal Agentic Framework**: `ChartAgent` uses GPT-4o (`gpt-4o-2024-08-06`) as the base MLLM. With its plug-and-play design, `ChartAgent` benefits from advances in both perception tools and MLLM reasoning, enabling seamless integration and sustained cumulative performance gains. We also experiment with other MLLMs to validate this generalization; see Sec. 5.2. `ChartAgent` is built on AutoGen (Wu et al., 2024a), which enables tool orchestration; see App. M for the structured set of prompts. After each ReAct cycle, `ChartAgent` evaluates the updated multimodal state s_{t+1} and decides whether to continue or terminate with a final answer. If satisfactory results cannot be achieved after multiple iterations, the agent gracefully falls back to direct MLLM reasoning (see Sec. 5.3 for evaluation). The maximum number of ReAct iterations is set to 15. Qualitative illustrations of agent trajectories are provided in App. J, with further implementation details in App. G.

4 Experimental Protocol and Details

Datasets We benchmark on two widely used datasets: **ChartBench** (Xu et al., 2023), which spans 9 chart categories and 42 subtypes, including standard charts (bar, line, pie) and complex ones (area, radar, box, scatter, node, and combinations), with 3,800 chart–QA pairs (76.2% unannotated). We evaluate two QA types: (1) *Numeric QA*, requiring precise value extraction, and (2) *Relationship QA*, involving structural reasoning (e.g., connectivity in graphs), with 96.7% numeric QA. **ChartX** (Xia et al., 2024), which covers 18 chart types, ranging from standard to domain-specific formats (e.g., treemaps, heatmaps, candlestick charts), with 1,152 chart–QA pairs (61.7% unannotated). The questions span (1) *Numeric QA*, and (2) *Value Comparison / Global Perception QA*, which involves reasoning over relative or extremum-based patterns, with 71.9% numeric QA. Both benchmarks are visually grounded, requiring models to reason about chart logic (e.g., bar heights, pie-slice areas) beyond OCR. Their high proportion of unannotated charts and numeric QA makes them particularly well-suited for evaluating complex visual reasoning. See App. C and D for dataset details.

Baselines We evaluate against 42 baseline models to ensure a comprehensive comparison: **(A) Proprietary MLLMs:** GPT-4o, GPT-4o-mini, Claude 3 Haiku, Gemini 1.5; **(B) Open-Weight General-Purpose MLLMs:** BLIP-2, CogAgent, CogVLM, DeepSeek-VL2, DocOwl1.5, InstructBLIP, InternVL3, LLaMA-3.2, LLaVA-1.6/1.5/OneVision, mPLUG-Owl3, Phi-3 Vision, Pixtral, Qwen2-VL, Qwen-VL-Chat, SmolVLM, SPHINX-V, VisualGLM; **(C) Chart-Specific MLLMs:** ChartGemma, ChartInstruct, ChartLLaMA, ChartVLM, DePlot, MatCha, OneChart, TinyChart, UniChart. *Concurrent Works:* We additionally include recently released models whose knowledge cutoffs are later than the dataset release or whose launch dates are concurrent with ours: GPT-o3/o4-mini/4.1/5/5-mini, Gemini 2.0 Flash, Claude 3.7 Sonnet/3.5 Sonnet/3.5 Haiku, and Mistral. We compare zero-shot and Chain-of-Thought (CoT) prompting; see App. M.2 for the corresponding prompts. Further details in App. E and Table 4.

Evaluation Metrics We use accuracy as the primary evaluation metric, computed via a two-step procedure. First, GPT-4o standardizes both the model's response and the ground truth—stripping units (e.g., "M" for million, "B" for billion), converting scales, removing symbols, and formatting numbers consistently (see App. H). If responses are numeric, we then apply an arithmetic correctness check with a strict 5% relative error tolerance; for non-numeric responses, we perform an exact string match after standardization. Prior work often uses the LLM-as-a-Judge paradigm (Masry et al., 2023, 2022; Xia et al., 2024; Xu et al., 2023), but we find it suboptimal for numerically precise answers under a 5% tolerance, as LLMs may inconsistently enforce thresholds or miss small deviations (see

Table 1: **Comparison of accuracy (%).** **Red:** Best, Blue: Second best. All values correspond to the highest performance achieved across zero-shot and CoT prompting styles for each MLLM. Ann./Unann. denote Annotated and Unannotated charts. RL QA: Relationship QA; VC/GC QA: Value Comparison & Global Conception QA.

Model	Chart Types		Question Types		Overall
	Ann.	Unann.	Numeric QA	RL QA	Avg. ↑
Proprietary Multimodal Large Language Models					
GPT 4o (Hurst et al., 2024)	**94.33**	36.15	52.50	**91.00**	54.53
GPT 4o-mini (GPT, 2024)	84.83	25.19	41.50	89.50	44.03
Claude 3 Haiku (Anthropic, 2024a)	84.58	26.04	42.94	73.00	44.53
Gemini 1.5 (Team et al., 2024b)	89.72	27.27	46.69	53.85	47.08
Open-weights Multimodal Large Language Models					
BLIP-2 (Li et al., 2023)	3.67	2.92	3.11	4.00	3.16
CogAgent (Hong et al., 2023)	69.92	11.62	30.28	27.00	30.03
CogVLM (Wang et al., 2023)	64.83	11.62	29.03	21.50	28.42
DeepSeek-VL2 (Wu et al., 2024c)	90.75	30.31	50.28	33.50	49.39
DocOwl1.5 (Hu et al., 2024a)	67.50	23.58	37.06	44.50	37.45
InstructBLIP (Dai et al., 2023)	3.92	5.92	4.22	24.50	5.29
InternVL3 (Zhu et al., 2025)	72.67	30.92	43.39	57.00	44.11
LLama3.2 (Grattafiori et al., 2024)	87.58	36.38	52.22	50.00	52.11
Llava1.6 (Liu et al., 2024b)	35.58	9.92	16.69	42.00	18.03
Llava1.5 (Liu et al., 2023c)	26.75	7.00	13.06	16.50	13.24
LlaVA-OneVision (Li et al., 2024)	13.25	10.50	9.94	37.00	11.37
mPLUG-Owl3 (Ye et al., 2024)	31.08	12.65	16.92	46.50	18.47
Phi3-vision (Abdin et al., 2024)	86.92	40.77	55.89	52.00	55.32
Pixtral (Agrawal et al., 2024)	66.58	28.73	39.53	63.50	40.50
Qwen2-VL (Wang et al., 2024a)	78.42	43.50	52.94	83.00	54.53
Qwen-VL-Chat (Bai et al., 2023)	27.17	6.54	12.61	21.00	13.05
SmolVLM (Marafioti et al., 2025)	47.75	14.46	23.14	58.00	24.97
SPHINX-V (Lin et al., 2025)	35.91	12.30	18.08	0.5	19.76
VisualGLM (GLM et al., 2024)	4.83	7.65	3.92	58.00	6.76
Chart-related Models					
ChartGemma (Masry et al., 2025)	75.92	22.42	39.56	35.00	39.32
ChartInstruct (Masry et al., 2024)	55.17	20.19	31.75	22.00	31.24
ChartLlama (Han et al., 2023)	38.25	11.42	18.81	39.50	19.89
ChartVLM (Xia et al., 2024)	61.00	23.92	36.97	11.50	35.63
DePlot (Liu et al., 2023a)	70.08	28.15	39.33	78.50	41.39
MatCha (Liu et al., 2023b)	59.50	9.69	25.86	17.50	25.42
OneChart (Chen et al., 2024)	56.78	26.81	35.22	62.76	36.81
TinyChart (Zhang et al., 2024)	77.33	32.77	47.86	28.50	46.84
UniChart (Masry et al., 2023)	53.50	15.96	27.44	34.50	27.82
Multimodal Agentic Framework (Ours)					
ChartAgent	**94.33**	**60.81**	**70.91**	**91.00**	**71.39**

(a) **ChartBench** (76.2% unannotated charts; 96.7% numeric QA)

Model	Chart Types		Question Types		Overall
	Ann.	Unann.	Numeric QA	VC/GC QA	Avg. ↑
Proprietary Multimodal Large Language Models					
GPT 4o	**84.84**	39.44	52.05	**69.14**	56.86
GPT 4o-mini	71.95	33.94	42.51	63.89	48.52
Claude 3 Haiku	63.57	25.77	35.99	51.23	40.28
Gemini 1.5	68.09	31.41	40.22	58.95	45.48
Open-weights Multimodal Large Language Models					
BLIP-2	1.13	1.69	0.72	3.40	1.48
CogAgent	46.15	24.93	27.05	48.46	33.07
CogVLM	46.38	24.23	24.28	54.32	32.73
DeepSeek-VL2	66.74	35.63	43.84	57.10	47.57
DocOwl1.5	42.53	24.37	26.81	42.90	31.34
InstructBLIP	10.41	8.87	7.37	14.81	9.46
InternVL3	65.84	36.62	44.20	57.10	47.83
LLama3.2	78.51	39.86	50.36	65.74	54.69
Llava1.6	26.24	18.17	16.55	33.33	21.27
Llava1.5	18.55	14.51	10.63	29.94	16.06
LlaVA-OneVision	20.14	12.82	13.89	20.06	15.62
mPLUG-Owl3	23.98	18.31	14.49	35.80	20.49
Phi3-vision	59.95	41.69	41.06	68.21	48.70
Pixtral	64.93	38.17	41.55	66.05	48.44
Qwen2-VL	76.24	42.96	51.81	65.74	55.73
Qwen-VL-Chat	24.66	20.42	11.59	48.77	22.05
SmolVLM	28.51	22.11	19.93	36.42	24.57
SPHINX-V	27.37	20.70	14.49	45.67	23.26
VisualGLM	9.28	13.10	4.47	29.94	11.63
Chart-related Models					
ChartGemma	45.93	28.87	27.54	55.56	35.42
ChartInstruct	27.38	17.75	20.29	24.38	21.44
ChartLlama	30.54	21.55	18.72	41.05	25.00
ChartVLM	46.83	29.01	35.75	36.11	35.85
DePlot	60.63	34.51	41.30	52.78	44.53
MatCha	28.28	17.04	18.24	29.32	21.35
OneChart	54.48	37.14	41.61	51.50	44.33
TinyChart	57.01	33.38	36.11	58.64	42.45
UniChart	24.66	18.87	16.06	33.95	21.09
Multimodal Agentic Framework (Ours)					
ChartAgent	**84.84**	**44.16**	**55.93**	**69.14**	**59.69**

(b) **ChartX** (61.7% unannotated; 71.9% numeric QA)

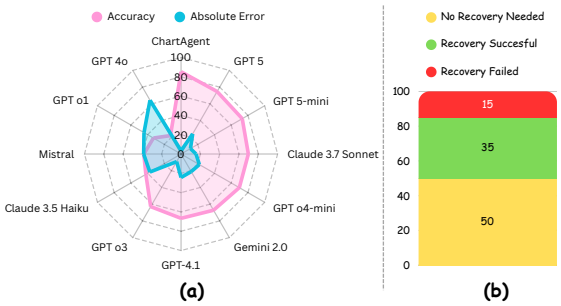


Figure 3: **(a) Left: `ChartAgent` vs. concurrent works:** overall accuracy (↑) and average absolute error (↓). **(b) Right: Effectiveness of visual self-verification:** enabled 70% successful recoveries when invoked.

App. K.5). See App. M.3 for evaluation prompts.

5 Results and Analysis

5.1 Performance

Comparison to State-of-the-art Table 1 presents a comparative analysis of `ChartAgent` against 32 baselines on the ChartBench and ChartX benchmarks, stratified by annotation status and QA type. `ChartAgent` *consistently outperforms all competing methods*, showing particularly strong gains on unannotated charts and numeric QA—the dominant categories across both datasets. On ChartBench, `ChartAgent` achieves 71.39% overall accuracy, a +16.07% absolute gain over the second-best model (Phi-3 Vision), including 60.81% on unannotated charts (+17.31% over Qwen2-VL) and 70.91% on numeric QA (+15.02% over Phi-3 Vision). A similar trend is observed on ChartX, where `ChartAgent` attains 59.69% overall accuracy (+2.83% absolute gain over GPT-4o), with top scores on unannotated (44.16%) and numeric QA (55.93%). Furthermore, Figure 3(a) and App. Table 14 present results comparing `ChartAgent` with 10 additional concurrent works on a newly curated dataset designed to ensure fair comparison and mitigate potential data leakage (see App. K.6). `ChartAgent` *outperforms all concurrent models* by a significant margin, achieving a +10.48% absolute accuracy gain over the second-best model (GPT-5) and a 5.72-point reduction in average absolute error relative to GPT-o3. Overall, these results establish **`ChartAgent` as the new SoTA in Chart VQA**, with major gains in numeric QA on unannotated charts, highlighting the value of visually

Table 2: **Accuracy on unannotated charts (%) by chart type. Red:** Best, Blue: Second best. **Abbreviations:** Over: Overlay | Stack: Stacked | Mul: Multi | Sing: Single | Hor: Horizontal | Vert: Vertical | B-L: Bar-Line | L-L: Line-Line | Dir: Directed | Undir: Undirected | Combo: Combination. See App. D for examples of each chart type.

Model	Area		Horizontal Bar			3D Bar		Vertical Bar			Box			Combo		Line		Node		Pie			Radar			Scatter	Avg. ↑
	Over	Stack	Mul	Sing	Stack	Mul	Stack	Mul	Sing	Stack	Hor	Vert	Stock	B-L	L-L	Mul	Sing	Dir	Undir	Mul	Ring	Sector	Mul	Fill	Sing	3D	
Proprietary Multimodal Large Language Models																											
GPT 4o	21.0	18.0	24.0	59.0	10.0	20.0	6.0	38.0	73.0	12.0	20.0	26.0	63.0	35.0	41.0	37.0	75.0	**91.0**	**91.0**	3.0	32.0	34.0	22.0	20.0	6.0	63.0	36.15
Gemini 1.5	5.0	4.0	28.0	52.0	7.0	14.0	4.0	39.05	49.0	5.0	13.0	18.0	24.0	28.0	5.0	7.0	**91.0**	48.0	59.26	1.0	14.0	29.52	1.0	7.0	0.0	45.0	27.27
Open-weights Multimodal Large Language Models																											
DeepSeek-VL2	29.0	11.0	25.0	57.0	8.0	36.0	8.0	58.0	82.0	13.0	11.0	3.0	51.0	46.0	48.0	51.0	8.0	31.0	36.0	0.0	6.0	15.0	13.0	21.0	5.0	44.0	30.31
InternVL3	25.0	16.0	45.0	80.0	19.0	38.0	1.0	44.0	80.0	16.0	16.0	23.0	60.0	27.0	24.0	30.0	56.0	62.0	52.0	0.0	2.0	9.0	**24.0**	24.0	6.0	25.0	30.92
LLama3.2	46.0	21.0	58.0	91.0	11.0	31.0	4.0	71.0	89.0	10.0	6.0	6.0	49.0	42.0	46.0	63.0	87.0	42.0	58.0	5.0	4.0	25.0	8.0	17.0	10.0	46.0	36.38
Phi3-vision	27.0	37.0	43.0	78.0	8.0	**40.0**	7.0	86.0	92.0	30.0	9.0	15.0	48.0	31.0	55.0	66.0	84.0	39.0	51.0	2.0	14.0	21.0	11.0	**26.0**	**66.0**	**73.0**	40.77
Pixtral	26.0	10.0	25.0	51.0	6.0	30.0	5.0	39.0	89.0	10.0	16.0	29.0	39.0	19.0	24.0	17.0	32.0	68.0	59.0	2.0	21.0	28.0	13.0	9.0	8.0	72.0	28.73
Qwen2VL	**57.0**	18.0	**87.0**	**97.0**	17.0	**40.0**	7.0	**94.0**	**97.0**	24.0	13.0	4.0	**64.0**	37.0	46.0	**80.0**	85.0	80.0	86.0	1.0	12.0	9.0	9.0	11.0	9.0	47.0	43.50
Chart-related Models																											
DePlot	18.0	2.0	43.0	74.0	13.0	34.0	9.0	66.0	78.0	7.0	20.0	20.0	0.0	48.0	45.0	14.0	63.0	84.0	73.0	4.0	3.0	5.0	2.0	2.0	3.0	2.0	28.15
TinyChart	32.0	22.0	71.0	88.0	13.0	37.0	**15.0**	76.0	82.0	21.0	2.0	3.0	4.0	46.0	50.0	51.0	**91.0**	22.0	35.0	1.0	20.0	21.0	10.0	8.0	4.0	27.0	32.77
Multimodal Agentic Framework (Ours)																											
`ChartAgent`	30.0	**38.0**	79.0	76.0	**82.0**	20.0	6.0	88.0	88.0	**76.0**	**89.0**	**83.0**	**64.0**	**67.0**	**65.0**	63.0	81.0	**91.0**	**91.0**	**18.0**	**94.0**	**80.0**	22.0	20.0	6.0	64.0	**60.81**

grounded agentic reasoning for charts.

Performance by Chart Type Table 2 compares `ChartAgent` with the top-10 baselines on unannotated charts, stratified by chart type on ChartBench (see App. K.1 for the full table and ChartX results). On ChartBench, `ChartAgent` achieves the largest gains on Bar (particularly horizontal and stacked variants, up to +65%), Box (up to +69%), Combination (Bar-Line, Multi-Line, up to +23%), and Pie (Ring, Sector, up to +62%) charts, while performance is weaker on 3D and Radar charts due to challenges in handling slanted, depth-distorted axes and radial coordinate structures. See App. J for qualitative examples and trajectories across chart types. Overall, these results underscore **`ChartAgent`'s robustness across a wide range of chart types**.

5.2 Effectiveness of `ChartAgent`

Performance Across Visual and Reasoning Complexity Levels We analyze `ChartAgent`'s performance across difficulty levels, stratified by (1) the *visual complexity of charts* and (2) the *reasoning complexity of chart–QA pairs*, each categorized into three levels: Easy, Medium, and Hard. See App. I for details and statistics, and App. M.4 for corresponding prompts. Figure 4(a) compares `ChartAgent` with the top-10 baselines on unannotated charts, stratified by these complexity levels on ChartBench (see App. K.4 for full results). All models show a consistent decline from Easy to Hard across both dimensions, confirming that visual clutter and multi-step reasoning increase Chart VQA difficulty. `ChartAgent` *achieves the best performance at all levels except visually Hard*, with notable gains on visually Easy (+18%) and Medium (+20.1%) charts, and reasoning Easy (+21.2%) and Medium (+20.8%) tasks. Visually Hard charts (17.9%) remain challenging due to 3D, radar, and overlapping structures that obscure segment boundaries and axis references. However, on reasoning Hard tasks involving multi-step numerical reasoning, `ChartAgent` still delivers a +6.9% gain. These results demonstrate **`ChartAgent`'s strong generalization across varying visual and reasoning complexities** in chart–QA pairs.

Plug-and-Play Generalization Across MLLMs `ChartAgent` follows a plug-and-play design, enabling seamless integration with any MLLM to provide chart-specialized, visually grounded reasoning. To assess generalization beyond GPT-4o as the base MLLM, we evaluate `ChartAgent` with three additional models: GPT-4o-mini, Claude 3 Haiku, and Pixtral. Figure 4(b) compares the performance of `ChartAgent`+Base MLLM versus the Base MLLM alone on unannotated and numeric Chart VQA. `ChartAgent` *consistently outperforms its corresponding base models*, yielding absolute accuracy gains of +26.7% on GPT-4o, +23.9% on GPT-4o-mini, +28.4% on Claude 3 Haiku, and +12.2% on Pixtral. Thus, **`ChartAgent` serves as an effective plug-and-play framework that enhances performance across diverse MLLMs**, demonstrating both robustness and generalization.

5.3 Additional Analysis

Effectiveness of Visual Self-Verification and Recovery We evaluated `ChartAgent`'s ability to detect unsatisfactory tool outputs and recover using its visual self-verification mechanism. Figure 3(b) and App. Table 15 summarize these results. Across 30 randomly sampled trajectories from ChartBench, tool outputs were correct and required no recovery in 50% of cases. In the remaining 50%, `ChartAgent` correctly flagged unsatisfactory outputs and triggered its self-verification mechanism, *recovering successfully 70% of the time* and failing 30%, with the latter contributing to a 15%

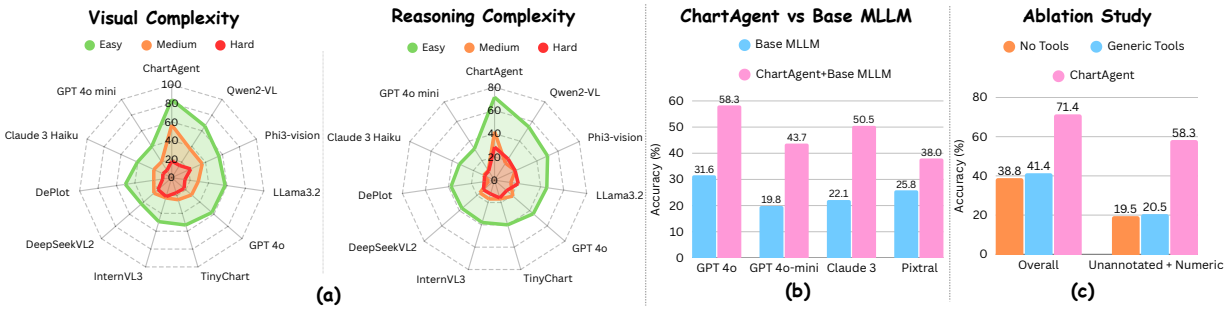


Figure 4: **Analysis of `ChartAgent` Performance. (a) Left:** Stratified by visual complexity of charts and reasoning complexity of chart–QA pairs on unannotated charts, compared with top-10 SoTA. **(b) Middle:** `ChartAgent` performance on unannotated+numeric chartQA when instantiated with different base MLLMs. **(c) Right:** Ablation study comparing `ChartAgent` with ReAct using no tools and ReAct with natural image–based generic tools.

overall error rate attributable to unresolved tool-level failures. Thus, **`ChartAgent`'s visual self-verification mechanism is both frequently invoked and often effective**, enhancing its robustness in the presence of imperfect tool outputs.

Ablation Study Prior frameworks for visually grounding MLLMs primarily focus on natural images and rely on generic tools such as cropping and zooming (Zheng et al., 2025; Su et al., 2025; Jegham et al., 2025; Hu et al., 2024b; Gupta and Kembhavi, 2023; Surís et al., 2023). While effective for object localization or text spotting, these tools lack the fine-grained capabilities required for structured, quantitative reasoning in charts. We compare three ReAct-style agents, all using GPT-4o as the base MLLM with visual self-verification: (i) ReAct (No Tools), (ii) ReAct + Natural Image Tools, with generic natural-image operations, and (iii) `ChartAgent`. Figure 4(c) shows that `ChartAgent` *outperforms both variants* by +32.6% over ReAct (No Tools) and +30.0% over ReAct + Image Tools overall, and by +38.8% and +37.8% respectively on the unannotated + numeric subset. These findings highlight the **limitations of generic tools and the necessity of chart-specialized visual grounding**. See App. K.3 for further details.

Fallback Behavior and Common Triggers We conducted a manual analysis of 30 randomly selected ChartBench trajectories (unannotated, numeric QA) to understand when and why `ChartAgent` reverts to the base MLLM. The *fallback rate was relatively low (below 10%)* and was typically triggered by: (1) bar charts with negative or axis-inconsistent bar-height estimates; (2) OCR tools returning None for legends or axis labels; and (3) edge-point detection or interpolation tools producing empty or axis-inconsistent outputs. In such cases, the agent identified tool-based reasoning as unreliable and reverted to the base MLLM, **a rare but effective fail-safe mechanism** that helps maintain robustness. See App. K.8 for further details.

5.4 Failure Mode Analysis

We conducted a failure mode analysis to identify common errors in `ChartAgent`, which fall into two main categories: **(1) Perception-based failures.** These stem from visual misinterpretations such as: (1.1) *OCR obstruction* from overlays or dense elements; (1.2) *Poor color contrast* (e.g., white text on yellow background); (1.3) *Legend occlusion* over key regions; (1.4) *Element invisibility* where lines or markers blend with background; (1.5) *Segmentation errors* caused by axis lines overlapping chart elements; (1.6) *Overlapping series* obscuring category distinctions; and (1.7) *Axis interpretation issues* in 3D or multi-axis charts with distorted or inconsistent scales across multiple axis. **(2) Reasoning-based failures.** (2.1) *Incorrect tool choice* (e.g., using area instead of height); (2.2) *Ambiguous queries* (e.g., missing denominators in multi-ring pies); and (2.3) *Label duplication* across hierarchy levels (e.g., "Netflix" as both parent and child). See App. L and Figures 21a, 21b for details.

6 Conclusion

We introduced `ChartAgent`, a novel multimodal agentic framework for visually grounded reasoning in charts. Inspired by human cognitive strategies of iterative reasoning and annotation-based chart comprehension, `ChartAgent` employs a multi-turn, tool-augmented interaction loop to achieve SoTA performance on well-established benchmarks spanning 40+ chart types, surpassing 40+ baselines with particularly strong gains on unannotated charts and numeric QA. Comprehensive analyses demonstrate its robustness across varying visual and reasoning complexity levels, its plug-and-play generalization across MLLMs, and the effectiveness of each agent component, supported by a failure mode analysis.

7 Limitations and Broader Perspective

Limitations and future work: The current approach focuses on question answering but can be extended to additional tasks such as data extraction, summarization, description, and fact-checking. Evaluation has so far been limited to single charts without broader context; future work can explore charts within slides or multi-chart settings. The in-context learning (ICL) examples are purely textual rather than multi-modal, integrating visual ICL (e.g., image-based examples) could enhance accuracy but would also increase context length, introducing a trade-off between richer supervision and computational efficiency. Future work should systematically examine this balance.

Another limitation lies in inference time and monetary cost when using large proprietary models such as OpenAI or Claude, as detailed in Appendices K.9 and K.10. The additional latency stems from the agentic design involving iterative reasoning, visual tool executions, and verification loops. Despite this overhead, the accuracy gains, especially on unannotated charts and numeric QA, justify the cost in precision-critical applications.

Finally, the vision tools are manually designed but generalize across 40+ chart types since they operate at the component level. Future work will explore automatic, on-the-fly tool construction. Some errors also arise from ambiguous queries; enabling the agent to detect and request clarification for such cases is an important direction for improvement.

Broader perspective: Prior work has highlighted the new and unpredictable risks associated with using automated agents in sensitive contexts (Wright, 2024). We advise against using this framework or MLLM agents to automate critical chart- or image-related tasks without human oversight. Additionally, the resources accompanying this study will be responsibly released for research purposes only.

Datasets and code: The benchmarks used in this study are publicly available and were curated by previous research. Specifically, we include the following datasets: ChartBench (Xu et al., 2023), ChartX (Xia et al., 2024), and ChartQA-unannotated (Islam et al., 2024). We abide by their terms of use. The implementation code is available at `https://anonymous.4open.science/r/ChartAgent`. To support reproducibility and facilitate future research, we will make the resources associated with this study available upon acceptance.

References

AutoGPT. https://github.com/Significant-Gravitas/Auto-GPT. Accessed: 2025-05-01.

CrewAI. https://github.com/crewAIInc/crewAI. Accessed: 2025-05-01.

EasyOCR. https://github.com/JaidedAI/EasyOCR. Accessed: 2025-05-01.

LangChain. https://github.com/langchain-ai/langchain. Accessed: 2025-05-01.

LangGraph. https://github.com/langchain-ai/langgraph. Accessed: 2025-05-01.

Tesseract OCR. https://github.com/tesseract-ocr/tesseract. Accessed: 2025-05-01.

2024. GPT4o-mini. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/. Accessed: 2025-05-01.

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Qin Cai, Martin Cai, Caio César Teodoro Mendes, Weizhu Chen, and 96 others. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *Preprint*, arXiv:2404.14219.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Pravesh Agrawal, Szymon Antoniak, Emma Bou Hanna, Baptiste Bout, Devendra Chaplot, Jessica Chudnovsky, Diogo Costa, Baudouin De Monicault, Saurabh Garg, Theophile Gervet, and 1 others. 2024. Pixtral 12b. *arXiv preprint arXiv:2410.07073*.

Jaided AI. 2020. Easyocr. https://github.com/JaidedAI/EasyOCR.

Anthropic. 2024a. The claude 3 model family: Opus, sonnet, haiku.

Anthropic. 2024b. Model card addendum: Claude 3.5 haiku and upgraded claude 3.5 sonnet. Accessed: 2025-07-09.

Anthropic. 2025. Claude 3.7 sonnet system card. Accessed: 2025-07-09.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*.

Jinyue Chen, Lingyu Kong, Haoran Wei, Chenglong Liu, Zheng Ge, Liang Zhao, Jianjian Sun, Chunrui Han, and Xiangyu Zhang. 2024. Onechart: Purify the chart structural extraction via one auxiliary token. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 147–155.

Jawad Chishtie, Iwona Anna Bielska, Aldo Barrera, Jean-Sebastien Marchand, Muhammad Imran, Syed Farhan Ali Tirmizi, Luke A Turcotte, Sarah Munce, John Shepherd, Arrani Senthinathan, and 1 others. 2022. Interactive visualization applications in population health and health services research: systematic scoping review. *Journal of medical Internet research*, 24(2):e27534.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. 2023. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Preprint*, arXiv:2305.06500.

Team GLM, :, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, and 40 others. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *Preprint*, arXiv:2406.12793.

Google. 2025. Gemini 2.0 flash model card. Accessed: 2025-07-09.

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

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

Yucheng Han, Chi Zhang, Xin Chen, Xu Yang, Zhibin Wang, Gang Yu, Bin Fu, and Hanwang Zhang. 2023. Chartllama: A multimodal llm for chart understanding and generation. *Preprint*, arXiv:2311.16483.

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

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. Cogagent: A visual language model for gui agents. *Preprint*, arXiv:2312.08914.

Chiori Hori, Motonari Kambara, Komei Sugiura, Kei Ota, Sameer Khurana, Siddarth Jain, Radu Corcodel, Devesh Jha, Diego Romeres, and Jonathan Le Roux. 2025. Interactive robot action replanning using multimodal llm trained from human demonstration videos. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.

Anwen Hu, Haiyang Xu, Jiabo Ye, Ming Yan, Liang Zhang, Bo Zhang, Chen Li, Ji Zhang, Qin Jin, Fei Huang, and 1 others. 2024a. mplug-docowl 1.5: Unified structure learning for ocr-free document understanding. *arXiv preprint arXiv:2403.12895*.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. 2024b. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Muye Huang, Lai Han, Xinyu Zhang, Wenjun Wu, Jie Ma, Lingling Zhang, and Jun Liu. 2024. Evochart: A benchmark and a self-training approach towards real-world chart understanding. *arXiv preprint arXiv:2409.01577*.

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

Mohammed Saidul Islam, Raian Rahman, Ahmed Masry, Md Tahmid Rahman Laskar, Mir Tafseer Nayeem, and Enamul Hoque. 2024. Are large vision language models up to the challenge of chart comprehension and reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 3334–3368, Miami, Florida, USA. Association for Computational Linguistics.

Nidhal Jegham, Marwan Abdelatti, and Abdeltawab Hendawi. 2025. Visual reasoning evaluation of grok, deepseek janus, gemini, qwen, mistral, and chatgpt. *arXiv preprint arXiv:2502.16428*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. 2018. Dvqa: Understanding data visualizations via question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656.

Samira Ebrahimi Kahou, Vincent Michalski, Adam Atkinson, Ákos Kádár, Adam Trischler, and Yoshua Bengio. 2017. Figureqa: An annotated figure dataset for visual reasoning. *arXiv preprint arXiv:1710.07300*.

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

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905, Bangkok, Thailand. Association for Computational Linguistics.

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

F Li, H Zhang, P Sun, X Zou, S Liu, J Yang, C Li, L Zhang, and J Gao. Semantic-sam: Segment and recognize anything at any granularity. arxiv 2023. *arXiv preprint arXiv:2307.04767*.

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

Weifeng Lin, Xinyu Wei, Ruichuan An, Peng Gao, Bocheng Zou, Yulin Luo, Siyuan Huang, Shanghang Zhang, and Hongsheng Li. 2025. Draw-and-understand: Leveraging visual prompts to enable MLLMs to comprehend what you want. In *The Thirteenth International Conference on Learning Representations*.

Fangyu Liu, Julian Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhu Chen, Nigel Collier, and Yasemin Altun. 2023a. DePlot: One-shot visual language reasoning by plot-to-table translation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10381–10399, Toronto, Canada. Association for Computational Linguistics.

Fangyu Liu, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Yasemin Altun, Nigel Collier, and Julian Eisenschlos. 2023b. MatCha: Enhancing visual language pretraining with math reasoning and chart derendering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12756–12770, Toronto, Canada. Association for Computational Linguistics.

Fuxiao Liu, Xiaoyang Wang, Wenlin Yao, Jianshu Chen, Kaiqiang Song, Sangwoo Cho, Yaser Yacoob, and Dong Yu. 2024a. Mmc: Advancing multimodal chart understanding with large-scale instruction tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1287–1310.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023c. Improved baselines with visual instruction tuning.

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. 2024b. Llava-next: Improved reasoning, ocr, and world knowledge.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023d. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916.

Shilong Liu, Hao Cheng, Haotian Liu, Hao Zhang, Feng Li, Tianhe Ren, Xueyan Zou, Jianwei Yang, Hang Su, Jun Zhu, and 1 others. 2025. Llava-plus: Learning to use tools for creating multimodal agents. In *European Conference on Computer Vision*, pages 126–142. Springer.

Junyu Luo, Zekun Li, Jinpeng Wang, and Chin-Yew Lin. 2021. Chartocr: Data extraction from charts images via a deep hybrid framework. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 1917–1925.

Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Vaibhav Srivastav, Joshua Lochner, Hugo Larcher, Mathieu Morlon, Lewis Tunstall, Leandro von Werra, and Thomas Wolf. 2025. Smolvlm: Redefining small and efficient multimodal models. *Preprint*, arXiv:2504.05299.

Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2263–2279, Dublin, Ireland. Association for Computational Linguistics.

Ahmed Masry, Parsa Kavehzadeh, Xuan Long Do, Enamul Hoque, and Shafiq Joty. 2023. UniChart: A universal vision-language pretrained model for chart comprehension and reasoning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14662–14684, Singapore. Association for Computational Linguistics.

Ahmed Masry, Mehrad Shahmohammadi, Md Rizwan Parvez, Enamul Hoque, and Shafiq Joty. 2024. ChartInstruct: Instruction tuning for chart comprehension and reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10387–10409, Bangkok, Thailand. Association for Computational Linguistics.

Ahmed Masry, Megh Thakkar, Aayush Bajaj, Aaryaman Kartha, Enamul Hoque, and Shafiq Joty. 2025. ChartGemma: Visual instruction-tuning for chart reasoning in the wild. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pages 625–643, Abu Dhabi, UAE. Association for Computational Linguistics.

Nitesh Methani, Pritha Ganguly, Mitesh M Khapra, and Pratyush Kumar. 2020. Plotqa: Reasoning over scientific plots. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1527–1536.

MistralAI. 2025. Mistral small 3.

Srija Mukhopadhyay, Adnan Qidwai, Aparna Garimella, Pritika Ramu, Vivek Gupta, and Dan Roth. 2024. Unraveling the truth: Do vlms really understand charts? a deep dive into consistency and robustness. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 16696–16717.

Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, and 1 others. 2024. Pivot: Iterative visual prompting elicits actionable knowledge for vlms. In *Forty-first International Conference on Machine Learning*.

OpenAI. 2024. O1 system card. https://cdn.openai.com/o1-system-card-20241205.pdf. Accessed: 2025-10-06.

OpenAI. 2025a. Gpt-5 system card. https://cdn.openai.com/gpt-5-system-card.pdf. Accessed: 2025-10-06.

OpenAI. 2025b. Introducing gpt-4.1 in the api. Accessed: 2025-07-09.

OpenAI. 2025c. Openai o3 and o4-mini system card. Accessed: 2025-07-09.

Yasaman Razeghi, Ishita Dasgupta, Fangyu Liu, Vinay Venkatesh Ramasesh, and Sameer Singh. 2024. Plot twist: Multimodal models don't comprehend simple chart details. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 5922–5937, Miami, Florida, USA. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. 2024. Visual cot: Advancing multi-modal language models with a comprehensive dataset and benchmark for chain-of-thought reasoning. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Ray Smith. 2007. An overview of the tesseract ocr engine. In *ICDAR '07: Proceedings of the Ninth International Conference on Document Analysis and Recognition*, pages 629–633, Washington, DC, USA. IEEE Computer Society.

Archita Srivastava, Abhas Kumar, Rajesh Kumar, and Prabhakar Srinivasan. 2025. Enhancing financial vqa in vision language models using intermediate structured representations. *arXiv preprint arXiv:2501.04675*.

Zhaochen Su, Peng Xia, Hangyu Guo, Zhenhua Liu, Yan Ma, Xiaoye Qu, Jiaqi Liu, Yanshu Li, Kaide Zeng, Zhengyuan Yang, and 1 others. 2025. Thinking with images for multimodal reasoning: Foundations, methods, and future frontiers. *arXiv preprint arXiv:2506.23918*.

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

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, Soroosh Mariooryad, Yifan Ding, Xinyang Geng, Fred Alcober, Roy Frostig, Mark Omernick, Lexi Walker, Cosmin Paduraru, Christina Sorokin, and 1118 others. 2024a. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *Preprint*, arXiv:2403.05530.

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

Gaurav Verma, Rachneet Kaur, Nishan Srishankar, Zhen Zeng, Tucker Balch, and Manuela Veloso. 2024. Adaptagent: Adapting multimodal web agents with few-shot learning from human demonstrations. *arXiv preprint arXiv:2411.13451*.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024a. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *Preprint*, arXiv:2409.12191.

Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang,

Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. Cogvlm: Visual expert for pretrained language models. *Preprint*, arXiv:2311.03079.

Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, and 1 others. 2024b. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Webb Wright. 2024. Ai agents with more autonomy than chatbots are coming. some safety experts are worried.

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

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024a. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

Yifan Wu, Lutao Yan, Leixian Shen, Yunhai Wang, Nan Tang, and Yuyu Luo. 2024b. Chartinsights: Evaluating multimodal large language models for low-level chart question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12174–12200.

Zhiyu Wu, Xiaokang Chen, Zizheng Pan, Xingchao Liu, Wen Liu, Damai Dai, Huazuo Gao, Yiyang Ma, Chengyue Wu, Bingxuan Wang, Zhenda Xie, Yu Wu, Kai Hu, Jiawei Wang, Yaofeng Sun, Yukun Li, Yishi Piao, Kang Guan, Aixin Liu, and 8 others. 2024c. Deepseek-vl2: Mixture-of-experts vision-language models for advanced multimodal understanding. *Preprint*, arXiv:2412.10302.

Renqiu Xia, Bo Zhang, Hancheng Ye, Xiangchao Yan, Qi Liu, Hongbin Zhou, Zijun Chen, Min Dou, Botian Shi, Junchi Yan, and 1 others. 2024. Chartx & chartvlm: A versatile benchmark and foundation model for complicated chart reasoning. *arXiv preprint arXiv:2402.12185*.

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

Zhengzhuo Xu, Sinan Du, Yiyan Qi, Chengjin Xu, Chun Yuan, and Jian Guo. 2023. Chartbench: A benchmark for complex visual reasoning in charts. *arXiv preprint arXiv:2312.15915*.

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

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

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

Jiabo Ye, Haiyang Xu, Haowei Liu, Anwen Hu, Ming Yan, Qi Qian, Ji Zhang, Fei Huang, and Jingren Zhou. 2024. mplug-owl3: Towards long image-sequence understanding in multi-modal large language models. *Preprint*, arXiv:2408.04840.

Liang Zhang, Anwen Hu, Haiyang Xu, Ming Yan, Yichen Xu, Qin Jin, Ji Zhang, and Fei Huang. 2024. Tinychart: Efficient chart understanding with program-of-thoughts learning and visual token merging. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1882–1898.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v (ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning*.

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

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, Zhangwei Gao, Erfei Cui, Xuehui Wang, Yue Cao, Yangzhou Liu, Xingguang Wei, Hongjie Zhang, Haomin Wang, Weiye Xu, and 32 others. 2025. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *Preprint*, arXiv:2504.10479.

Appendix

Table of Contents

A Annotated vs. Unannotated Charts 14

B Related Work 14

C Datasets 16

D Chart Types Supported in ChartAgent 17

E Baselines 20

F Taxonomy of Tools in ChartAgent 21

G Implementation Details 23

H Examples of Response Standardization for Accuracy Evaluation 26

I Complexity Analysis 26

J Qualitative Analysis 28

K Expanded Discussion on Results 35

L Details on Failure Mode Analysis 41

M Prompts 44

A Annotated vs. Unannotated Charts

An ***annotated chart*** contains explicit textual annotations or shortcuts. For instance, in bar charts, exact values may be printed above or inside the bars; in pie charts, percentage labels may appear alongside slices. In some cases, answers to questions may even be embedded in the title or legend. Generally, an annotated chart includes values visibly placed near the relevant graphical elements, though the information may also appear elsewhere within the chart image, such as in captions or legends. These textual cues allow models like GPT-4o to directly extract information from the image, often producing correct answers without requiring complex visual reasoning.

In contrast, ***unannotated charts*** lack such explicit value indicators. Consequently, the model must infer values by interpreting graphical features—such as bar heights, pie slice angles, or positions along axes. These tasks demand fine-grained visual perception and structured reasoning, often exceeding the capabilities of general-purpose LLMs or MLLMs alone.

A.1 Examples

Figure 5 illustrates this distinction, showing representative examples of both annotated and unannotated charts from the datasets.

A.2 How ChartAgent Handles Annotated vs. Unannotated Charts

Given a chart, ChartAgent first classifies it as annotated or unannotated using an LLM-based orchestrator (e.g., GPT-4o). On a uniformly sampled subset, this classification step achieves 100% accuracy. ChartAgent dynamically adapts its execution pathway based on the annotation type. For *annotated charts*—where text extraction alone is sufficient—the agent directly forwards the query to the base model (e.g., GPT-4o), which already achieves over 90% accuracy (see Table 1). This approach ensures both high performance and computational efficiency. For *unannotated charts*, however, ChartAgent triggers its full ReAct-style loop. Here, the agent's iterative reasoning and specialized visual tool use become essential to accurately extract values and answer queries, as detailed in Section 3.

B Related Work

B.1 Chart Visual Question Answering

Chart visual question answering (Chart VQA) aims to automatically interpret visual charts to answer natural-language queries. Early datasets such as FigureQA (Kahou et al., 2017) and DVQA (Kafle et al., 2018) introduced synthetic charts designed to evaluate specific reasoning skills but lacked real-world diversity. This gap was subsequently addressed by more comprehensive datasets like PlotQA (Methani et al., 2020), ChartQA (Masry et al., 2022), and EvoChart (Huang et al., 2024), which incorporated complex, real-world charts coupled with natural-language queries. Recent benchmarks such as ChartBench (Xu et al., 2023), ChartX (Xia et al., 2024), and CharXiv (Wang et al., 2024b) have further expanded the complexity and diversity of tasks, covering a wide range of chart types and numeric-intensive queries. These benchmarks reflect a growing trend toward datasets that demand sophisticated visual comprehension combined with nuanced quantitative reasoning.

Advancements in chart-focused multimodal large language models (MLLMs) (Zhang et al., 2024; Masry et al., 2023; Han et al., 2023; Wu et al., 2024b; Mukhopadhyay et al., 2024; Liu

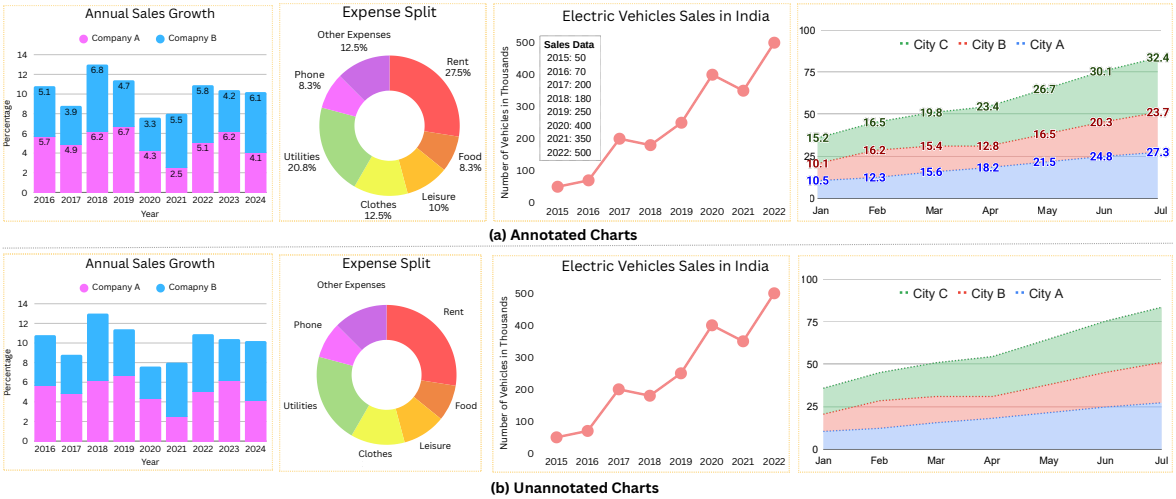


Figure 5: **Examples of annotated (top) vs. unannotated (bottom) charts.** An ***annotated chart*** contains explicit textual annotations or shortcuts, whereas an ***unannotated chart*** lacks such explicit value indicators. For instance, in the first column (top), the bar chart includes printed bar values, while in the corresponding bottom chart, the values must be inferred through visual interpretation.

et al., 2024a; Masry et al., 2024) have demonstrated notable progress by leveraging instruction-tuned datasets and vision-language alignment methods. Alternatively, ChartOCR (Luo et al., 2021) combines computer vision tools and rule-based techniques, such as keypoint detection and chart-specific rules, for enhanced chart understanding. However, recent studies (Xu et al., 2023; Razeghi et al., 2024; Islam et al., 2024) reveal persistent limitations, particularly in precise numerical interpretation tasks involving unannotated charts—visualizations lacking textual shortcuts such as numeric annotations or labels. In particular, (Xu et al., 2023) showed a significant performance drop when transitioning from annotated charts (containing textual cues) to unannotated charts, highlighting models' dependence on optical character recognition (OCR) rather than genuine visual reasoning. Addressing this limitation requires enhanced visual grounding capabilities that enable accurate interpretation and numerical reasoning directly from graphical elements (e.g., bar heights, segment areas).

Our approach specifically targets this challenge by enhancing MLLMs with modular, specialized vision tools tailored explicitly to the chart domain, thereby significantly improving visual reasoning and grounding in Chart VQA.

B.2 General-Purpose Multimodal LLMs and Visual Grounding

While recent chart-specific multimodal models have made notable progress, broader developments in general-purpose multimodal large language models (MLLMs)—such as GPT-4 (Achiam et al., 2023), GPT-4o (Hurst et al., 2024), Gemini (Team et al., 2023), and LLaVA (Liu et al., 2023d), Visual CoT (Shao et al., 2024)—have significantly advanced general visual reasoning and understanding across various tasks and domains. However, these general-purpose MLLMs also face challenges when tasks demand precise visual grounding and fine-grained interpretation of visual information.

To address these limitations, recent approaches have explored augmenting language and multimodal models with external tools or visual prompting. For instance, ToolFormer (Schick et al., 2023) integrates text-based language models with external APIs, demonstrating improved reasoning through external knowledge retrieval. Similarly, Visual ChatGPT (Wu et al., 2023) and MM-ReAct (Yang et al., 2023b) enhance text-only ChatGPT with vision expert tools for multimodal tasks. For MLLMs, ViperGPT (Surís et al., 2023) and VisProg (Gupta and Kembhavi, 2023) generate executable code via LLMs to perform sequences of tool invocations, though their execution follows a fixed plan without flexibility for dynamic adaptation based on intermediate tool outcomes. In contrast, methods like Visual Sketchpad (Hu et al., 2024b) explicitly incorporate intermediate visual results into iterative reasoning, enabling dynamic refinement of action plans based on observed outcomes.

Additionally, visual prompting methods such as Set-of-Marks (SoM) (Yang et al., 2023a) augment

input images with visual annotations (e.g., bounding boxes or segmentation masks), providing richer context to LLMs for informed reasoning. Inspired by SoM, our approach similarly presents the multimodal agent with explicit visualizations of intermediate tool outputs, enabling visual inspection and informed decision-making at each reasoning step.

Motivated by these advancements, our work extends multimodal LLM capabilities specifically into the chart domain, combining iterative reasoning, dynamic visual prompting, and modular external tools. Unlike fixed-sequence approaches, our framework enables adaptive replanning and precise visual grounding, effectively addressing complex chart interpretation tasks.

B.3 Agentic Frameworks

The concept of agents—entities capable of perception, cognition, and action—has long been foundational in artificial intelligence research. Traditional agents perceive their environment, reason about possible actions, and execute these actions to achieve specific goals. Recent advances in large language models (LLMs) have inspired a new generation of LLM-based agents, leveraging powerful reasoning capabilities and dynamic interactions with external tools. A notable example of aligning LLM reasoning explicitly with the agent paradigm is the ReAct framework (Yao et al., 2023), which organizes model interactions into iterative cycles of reasoning (cognition), action execution (action), and observing results (perception). This structured loop allows LLM-based agents to refine their decisions dynamically, closely mirroring traditional agent definitions.

Several software frameworks and platforms now support the practical implementation of LLM-based agents, enabling seamless integration of external tool usage within iterative reasoning loops. Examples include AutoGen (Wu et al., 2024a), CrewAI (cre), LangChain (Lan), LangGraph (lan), and AutoGPT (aut), each providing flexible infrastructures to orchestrate sophisticated LLM-driven workflows.

Extending this agentic paradigm into multimodal settings has further expanded agent capabilities across diverse applications. Multimodal agents effectively handle tasks in software engineering (Jimenez et al., 2024; Hong et al., 2024), robotics (Nasiriany et al., 2024), general vision-language reasoning (Liu et al., 2025; Yang et al., 2023b), and GUI navigation (Xie et al., 2024; Koh et al., 2024; Zheng et al., 2024; Verma et al., 2024). These frameworks dynamically combine visual perception with iterative LLM reasoning, adjusting action plans based on multimodal feedback. Chart VQA introduces unique challenges that specifically require chart-oriented perception and numeric reasoning capabilities.

Our proposed `ChartAgent` explicitly adopts the ReAct agentic framework (Yao et al., 2023), integrating iterative multimodal reasoning with carefully designed modular perception tools specifically tailored for chart understanding tasks. The practical implementation of our agent leverages AutoGen (Wu et al., 2024a), providing a flexible infrastructure for orchestrating dynamic interactions between the multimodal LLM and external tools, enabling effective iterative refinement and visual grounding.

C Datasets

To evaluate our agent's ability to understand charts, we design experiments that require complex visual reasoning, specifically focusing on question answering over *unannotated* charts, where accurate numerical interpretation and output precision are critical. We evaluate `ChartAgent` on two well-established and widely used chart QA benchmarks: ChartBench (Xu et al., 2023) and ChartX (Xia et al., 2024). These benchmarks are visually grounded—models must interpret the visual logic of the chart to answer questions, without relying solely on OCR. They are designed to assess chart comprehension and data reliability through complex reasoning, and the majority of their charts are unannotated (see Appendix A), making them ideal for testing visual understanding.

C.1 ChartBench

ChartBench (Xu et al., 2023) comprises charts from 9 major categories and 42 subcategories, with unannotated charts present across all 9 categories and over 75% of images being unannotated. It includes both regular chart types (line, bar, pie) and diverse, complex types such as area, box, radar, scatter, node, and combination charts (e.g., bar+line, bar+pie). The test set originally contained 2,100 images (50 per subcategory), but we discarded 4 subcategories with corrupted or incorrect ground-truth labels, yielding a final set of 1,900 images. We use two subsets of the ChartBench QA data: Numeric Question Answering (NQA) and

Value Extraction (VE), resulting in 3,800 image-QA pairs. ChartBench includes two primary types of questions: 1) *Numeric QA* — questions requiring precise numerical extraction (e.g., "What is the value of India in 2021?" or "How much more is A than B?"); 2) *Relationship QA* — questions involving relational understanding (e.g., "Is node A connected to node B?" or "Is node A directed toward node B?").

C.2 ChartX

ChartX (Xia et al., 2024) comprises charts from 18 categories, including regular types such as line, bar, and pie charts, as well as fine-grained and domain-specific charts such as ring charts, radar charts, box plots, 3D-bar charts, histograms, treemaps, rose charts, bubble charts, multi-axes charts, area charts, heatmaps, funnels, and candlestick charts. The dataset includes 1,152 image-question pairs in the test set, with more than 60% of the images being unannotated. ChartX includes two primary types of questions: 1) Numeric QA — questions that require precise numerical extraction; 2) Value Comparison and Global Perception QA — questions that require relative or extremum-based reasoning (e.g., identifying the highest, lowest, or most relevant entity), where exact values are not necessary. Examples of global perception questions include: "Which country has the highest GDP?", "Which region planted the most trees?", "Are there more trees planted in 2021 in region A or region B?"

It is important to note that **ChartX is a much harder dataset, both in terms of questions and chart samples**. The questions are more varied and open-ended; for example, "How many countries have CO_2 emissions greater than or equal to 350 million metric tons?" and "How many non-profits received donations in the range of 50K to 100K?" require computing all entries and then applying careful numeric filtering, which increases error susceptibility. The chart samples themselves are also more challenging: a significant fraction are occluded charts, where legends often overlap bars or chart elements of interest; many multi-axis plots involve three or more Y-axes; and in some cases grid lines are the same color as the bar or the box in box plots, making it difficult to distinguish regions of interest even after segmentation. Overall, ChartX presents a substantially more challenging testbed.

C.3 Dataset Statistics

Table 3 presents the chart type, annotation, and QA type distribution across the two evaluation datasets, ChartBench and ChartX. A key observation is the dominance of unannotated charts, which constitute over 76% of ChartBench and over 61% of ChartX. As discussed in Appendix A, such unannotated samples require visual extraction of values from chart elements rather than relying on textual annotations or shortcuts, thereby posing greater difficulty. Another important characteristic is the prevalence of numeric QA, comprising more than 94% in ChartBench and nearly 72% in ChartX. Taken together, these properties underscore that both datasets serve as rigorous testbeds for evaluating chart reasoning systems under visually demanding and numerically intensive conditions.

Note that we did not use the popular ChartQA (Masry et al., 2022) dataset, as all charts are annotated and MLLM performance on it already exceeds 85% due to strong OCR capabilities. We also excluded the CharXiv (Wang et al., 2024b) dataset, as it lacks numerically precise questions and primarily focuses on descriptive and reasoning-based queries. Thus, ChartBench and ChartX were selected for evaluation as they emphasize unannotated charts and require models to demonstrate true visual understanding and numerical reasoning beyond text extraction. See Appendix D for visualizations of the diverse chart types included in our benchmark datasets.

D Chart Types Supported in `ChartAgent`

`ChartAgent` supports a wide range of chart types across both the ChartBench and ChartX datasets. Specifically, ChartBench contains 9 major categories and 38 subcategories of charts (excluding 4 with corrupted or incorrect ground-truth labels), while ChartX comprises 18 types organized into three subcategories—general, fine-grained, and domain-specific. Figure 6 illustrates examples of each ChartX chart type, and Figure 7 presents the corresponding examples from the ChartBench dataset.

Table 3: **Dataset Statistics.** Chart type, annotation, and QA type distribution in the evaluation datasets.

(a) **ChartBench (3800 Image-QA pairs):** Over 75% unannotated charts; approximately 95% numeric QA.

Chart Type	% Annotated	% Unannotated	Regular Types			Extra (Diverse/Complex) Types					
			Line	Bar	Pie	Area	Box	Radar	Scatter	Node	Combination
ChartBench	23.80%	76.20%	11.90%	31.00%	11.90%	7.10%	7.10%	9.50%	7.10%	4.80%	11.90%

QA Type	% Numeric QA	% Non-Numeric QA
ChartBench	94.74%	5.26%

(b) **ChartX (1152 Image-QA pairs):** Over 60% unannotated charts; over 70% numeric QA.

Chart Type	% Annotated	% Unannotated	Regular Types			Extra (Diverse/Complex) Types												
			Line	Bar	Pie	Area	Box	Radar	Ring	3D-Bar	Histogram	Treemap	Rose	Bubble	Multi-axes	Heatmap	Funnel	Candlestick
ChartX	38.28%	61.72%	17.36%	17.36%	8.68%	4.34%	4.34%	4.34%	4.34%	4.34%	4.34%	4.34%	4.34%	4.34%	4.34%	4.51%	4.34%	4.34%

QA Type	% Numeric QA	% Non-Numeric QA
ChartX	71.88%	28.12%

Figure 6: **Chart types in the ChartX dataset**: 18 types organized into three subcategories—general, fine-grained, and domain-specific chart types, with the percentage of data in each subcategory indicated. Over 60% of the data is unannotated, making ChartX a robust testbed for visual reasoning in charts.

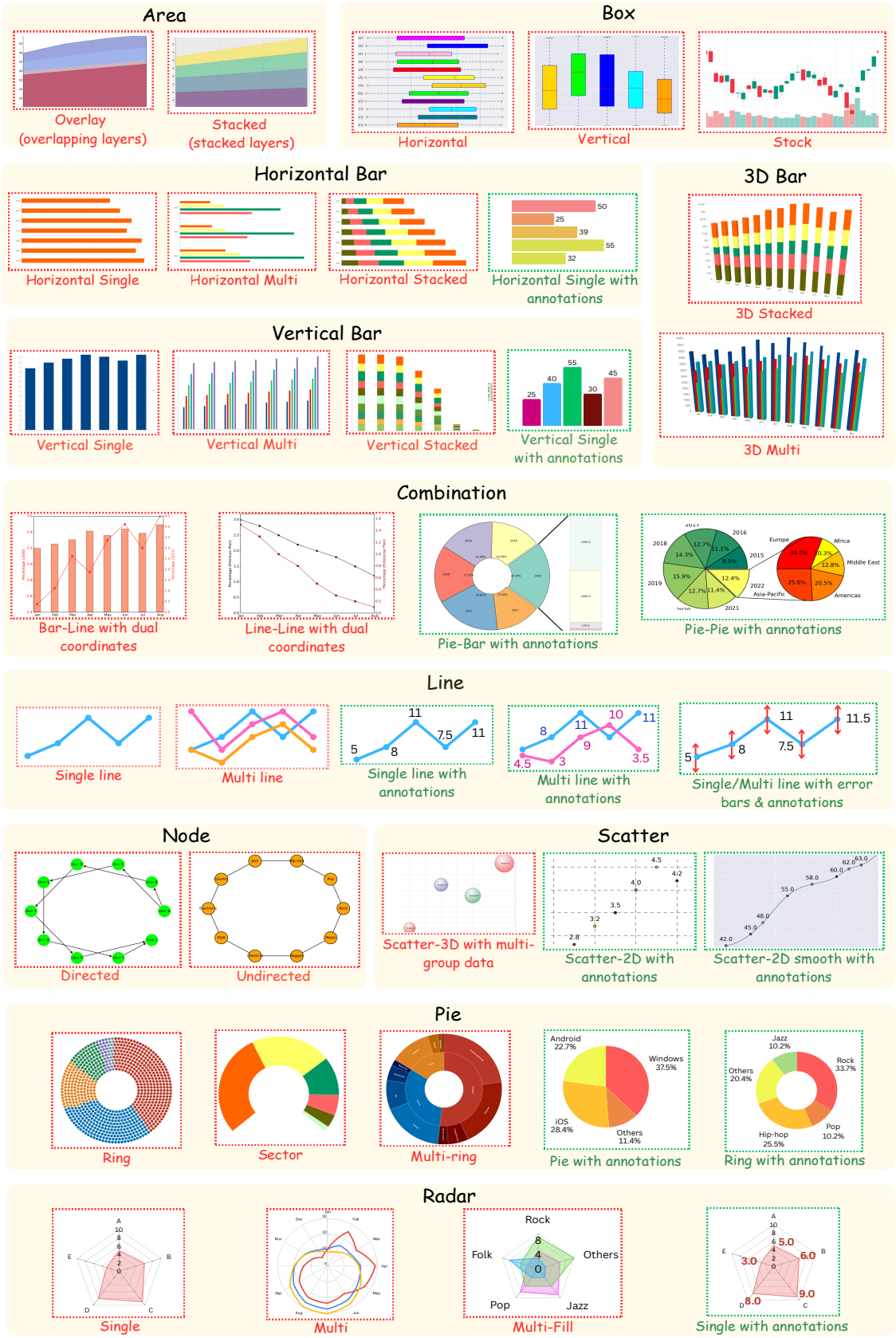


Figure 7: **Chart types in the ChartBench dataset**: 9 major types with 38 subtypes (excluding 4 subtypes with corrupted or incorrect ground-truth labels). Annotated subtypes are marked in green, and unannotated subtypes are marked in red. Over 75% of the data is unannotated, making ChartBench a robust testbed for visual reasoning in charts.

E Baselines

Table 4 summarizes the model architecture details of all baseline MLLMs compared in our experiments, including both proprietary and open-weight models—covering general-purpose as well as chart-specific open-weight MLLMs. See Appendix G.2 for implementation details and Appendix M.2 for prompts.

Table 4: **Model architectures of baseline MLLMs** considered in our experiments, including both proprietary and open-weight models—covering general-purpose and chart-based open-weight MLLMs. We report the model version (for proprietary models) or the underlying component architectures (for open-weight models), along with the name and parameter sizes of the vision encoder and language model (where applicable), and official access links. Concurrent works with knowledge cutoff dates after the release of our benchmark datasets (ChartBench, ChartX) are highlighted in orange.

Model	Version	Link
GPT 4o (Hurst et al., 2024)	`gpt-4o-2024-08-06`	OpenAI/gpt4o
GPT 4o-mini (GPT, 2024)	`gpt-4o-mini-2024-07-18`	OpenAI/gpt4o-mini
GPT 5 (OpenAI, 2025a)	`gpt-5-2025-08-07`	OpenAI/gpt5
GPT 5-mini (OpenAI, 2025a)	`gpt-5-mini-2025-08-07`	OpenAI/gpt5-mini
Gemini 1.5 flash (Team et al., 2024a)	`gemini-1.5-flash-002`	Google/gemini-1.5-flash
Claude 3 Haiku (Anthropic, 2024a)	`claude-3-haiku-20240307`	Anthropic/claude-3-haiku
GPT o1 (OpenAI, 2024)	`o1-2024-12-17`	OpenAI/o1
GPT o4-mini (OpenAI, 2025c)	`o4-mini-2025-04-16`	OpenAI/o4-mini
GPT o3 (OpenAI, 2025c)	`o3-2025-04-16`	OpenAI/o3
GPT o4-mini (OpenAI, 2025c)	`o4-mini-2025-04-16`	OpenAI/o4-mini
GPT 4.1 (OpenAI, 2025b)	`gpt-4.1-2025-04-14`	OpenAI/gpt4.1
Gemini 2.0 flash (Google, 2025)	`gemini-2.0-flash-001`	Google/gemini-2.0-flash
Claude 3.7 Sonnet (Anthropic, 2025)	`claude-3-7-sonnet-20250219`	Anthropic/claude-3.7-sonnet
Claude 3.5 Sonnet (Anthropic, 2024b)	`claude-3-5-sonnet-20240620`	Anthropic/claude-3.5-sonnet
Claude 3.5 Haiku (Anthropic, 2024b)	`claude-3-5-haiku-20241022`	Anthropic/claude-3.5-haiku

(a) Proprietary Multimodal Large Language Models

Model	Vision Encoder		Language Model		Link
	Name	Size	Name	Size	
General−Purpose MLLMs					
BLIP-2 (Li et al., 2023)	EVA-CLIP ViT	1.1B	OPT	2.7B	Salesforce/blip2-opt-2.7b
CogAgent (Hong et al., 2023)	EVA2-CLP-E	11B	Vicuna1.5	7B	THUDM/cogagent-vqa-hf
CogVLM (Wang et al., 2023)	EVA2-CLP-E	11B	Vicuna1.5	7B	THUDM/cogvlm-vqa-hf
DeepSeek-VL2 (Wu et al., 2024c)	SigLIP-SO400M	878M	DeepSeek MoE	16.4B	deepseek-ai/deepseek-vl2-small
DocOwl1.5-Chat (Hu et al., 2024a)	ViT/L	304M	Llama2	7B	mPLUG-DocOwl/DocOwl1.5
InstructBLIP (Dai et al., 2023)	BLIP2-QFormer	188M	Vicuna	7B	LAVIS/instrucblip
InternVL3 (Zhu et al., 2025)	InternViT	300M	Qwen2.5	1.5B	OpenGVLab/InternVL3-2B
LLama3.2 (Grattafiori et al., 2024)	ViT	630M	Llama3.1	8B	meta-llama/Llama-3.2-11B-Vision
Llava1.6 (Liu et al., 2024b)	CLIP ViT	304M	Mistral	6.74B	llava-hf/llava-v1.6-mistral-7b-hf
Llava1.5 (Liu et al., 2023c)	CLIP ViT	304M	Vicuna1.5	7B	liuhaotian/llava-v1.5-7b
LlaVA-OneVision (Li et al., 2024)	SigLIP	894M	Qwen2	494M	lmms-lab/llava-onevision-quen2-0.5b-ov
mPLUG-Owl3 (Ye et al., 2024)	SigLIP	400M	Qwen2	7B	mPLUG/mPLUG-Owl3-7B-240728
Phi3-vision (Abdin et al., 2024)	CLIP ViT	428M	Phi-3 Mini	3.8B	microsoft/Phi-3-vision-128k-instruct
Pixtral (Agrawal et al., 2024)	Pixtral-ViT	400M	Mistral-Nemo	12B	mistralai/Pixtral-12B-2409
Qwen2-VL (Wang et al., 2024a)	ViT	675M	QwenLM	7.6B	Qwen/Qwen2-VL-7B-Instruct
Qwen-VL-Chat (Bai et al., 2023)	ViT-bigG	1.9B	QwenLM	7.7B	Qwen/Qwen-VL-Chat
SmolVLM (Marafioti et al., 2025)	SigLIP	428M	SmolLM2	1.7B	HuggingFaceTB/SmolVLM-Instruct
SPHINX-V (Lin et al., 2025)	ViT-H SAM	636M	Llama2	13B	AFeng-x/Draw-and-Understand
VisualGLM (GLM et al., 2024)	BLIP2-QFormer	188M	ChatGLM	6.2B	THUDM/visualglm-6b
Mistral (MistralAI, 2025)	Mistral-small 22.2B				mistralai/Mistral-Small-Instruct-2409
Chart−related MLLMs					
ChartGemma (Masry et al., 2025)	PaliGemma 3B				ahmed-masry/chartgemma
ChartInstruct (Masry et al., 2024)	UniChart	201M	LLama2	7B	ahmed-masry/ChartInstruct-LLama2
ChartLlama (Han et al., 2023)	LLaVA1.5 13B				tingxueronghua/ChartLlama-code
ChartVLM (Xia et al., 2024)	Pix2Struct	282M	Vicuna1.5	7B	U4R/ChartVLM-base
DePlot (Liu et al., 2023a)	Pix2Struct 282M				google/deplot
MatCha (Liu et al., 2023b)	Pix2Struct 282M				google/matcha-chartqa
OneChart (Chen et al., 2024)	SAM-base	86M	OPT	125M	kppkkp/OneChart
TinyChart (Zhang et al., 2024)	TinyLLaVA 3.1B				mPLUG/TinyChart-3B-768
UniChart (Masry et al., 2023)	Donut-base 201M				vis-nlp/UniChart

(b) Open-weight Multimodal Large Language Models

F Taxonomy of Tools in ChartAgent

Table 5 provides a summary and description of the key vision and analytical tools used in ChartAgent.

Table 5: **Taxonomy of Tools in ChartAgent.** Summary of key vision and analytical tools used in ChartAgent.

Chart Type	Chart Tool	Description
Universal Tools		
All	annotate_legend	Detects legend coordinates, crops the legend, and annotates it with numeric labels. ***Returns the cropped and annotated legend image*** along with label mappings.
	get_marker_rgb	***Retrieves the dominant RGB color of a legend marker***, either by label (from an annotated legend image) or by associated text.
	clean_chart_image	***Detects and removes the title and legend*** (if present) from the chart image to avoid interference with downstream visual analysis such as OCR, segmentation, or edge detection.
	segment_and_mark	***Segments an input image*** using the specified model and applies post-processing to clean the masks. This includes a multi-step filtering pipeline that removes small, duplicate, composite, and background-dominated masks. ***Returns a labeled image with drawn contours and optional numbered labels***, along with a cleaned list of segmentation masks. Uses Segment Anything (ViT-H) as the default segmentation model (Kirillov et al., 2023).
	axis_localizer	***Localizes the specified axis*** (x-axis, left y-axis, or right y-axis) by detecting its numeric tick values and mapping them to corresponding pixel positions in the chart image. Uses Tesseract OCR (Smith, 2007) and EasyOCR (AI, 2020).
	interpolate_pixel_to_value	***Maps a pixel coordinate to its corresponding axis value*** using linear interpolation between known axis ticks and their pixel positions.
	arithmetic	***Performs a specified arithmetic operation*** between two numeric inputs. Supports operations such as addition, subtraction, multiplication, division, percentage, and ratio.
Chart-specific Tools		
Pie, Treemap	compute_segment_area	***Computes the area of a chart segment*** by: (1) counting discrete visual elements of a specified color, (2) counting pixels of a specified color, or (3) counting pixels within a segment identified by a specific label ID.
Bar, Combination	get_bar	***Detects and returns the bounding box of a bar*** in a chart image that matches a specified color and/or axis label. It segments bar regions using a model, filters by color if provided, locates the target axis label using OCR if specified, and selects the closest matching bar accordingly.
	compute_bar_height	***Computes the height or length of a bar*** in value space by mapping its pixel coordinates to axis values using OCR-based axis detection and localization.
Box	get_boxplot	***Detects and returns boxplot segments*** filtered by color, axis label, or segmentation indices. Handles both horizontal and vertical boxplot orientations and supports fuzzy matching for axis-aligned labels and approximate color filtering.
	compute_boxplot_entity	***Computes a statistical entity (e.g., max, min, median, Q1, Q3, range, or interquartile range) of a boxplot*** by mapping its pixel coordinates to value space using axis localization.
Line, Area, Scatter, Combination	get_edgepoints	***Computes edge points of a chart segment*** filtered by color, axis label, or segmentation indices. The edge is determined by scanning perpendicular to the center of the matched label. Supports both vertical and horizontal chart orientations and optionally *handles lineplot dots*. Useful for identifying segment bounds for downstream value extraction.
Radial Bar	get_radial	***Computes the coordinates for the radial bar segment*** of interest using either color-based filtering or segmentation mask labels.
	analyze_radial_geometry	Estimates the radial geometry of a radial bar chart for the segment of interest. ***Identifies the chart center, detects the outer circle representing the maximum value, and computes the maximum radial extent (i.e., radius) of the contour of interest.***
	estimate_radial_value	***Estimates the value of a radial segment in a radial bar chart*** by scaling its radial length relative to the outermost circle. The reference value for the outer circle is provided externally (e.g., by an LLM), with a default of 100.

These tools are categorized into two broad groups:

(1) **Universal tools**, which are applicable across all chart types. These include legend annotation (`annotate_legend`), axis localization (`axis_localizer`), marker color detection (`get_marker_rgb`), chart cleaning (`clean_chart_image`), segmentation (`segment_and_mark`), interpolation from pixel to value space (`interpolate_pixel_to_value`), and arithmetic operations (`arithmetic`).

(2) **Chart-specific tools**, which are designed for particular chart types. For example, pie and treemap charts use `compute_segment_area`; bar charts use `get_bar` and `compute_bar_height`; box plots use `get_boxplot` and `compute_boxplot_entity`; line, area, and scatter charts use `get_edgepoints`; and radial bar charts use `get_radial`, `analyze_radial_geometry`, and `estimate_radial_value`. For **combination charts** (e.g., bar+line or bar+pie), a combination of the relevant chart-specific tools is applied.

F.1 Underlying Models Powering `ChartAgent` Tools

While the specific function-calls are abstracted away from the agent, some of the universal tools above require the custom model:

(1) Semantic segmentation: `Segment Anything Model v1` is used by the `segment_and_mark` tool extract the chart contents (foreground) from the background, generate masked segments, and remove any unneeded content as well as duplicates. `Segment Anything` uses a (ViT-H) (Kirillov et al., 2023) model as an image encoder (641M parameters), alongside a prompt encoder, and a mask decoder.

(2) Optical Character Recognition: `Tesseract` (Smith, 2007) is used for quick OCR and localization tasks, such as extracting the chart x- and y- tick data in `axis_localizer` and legend information in `annotate_legend`. `Tesseract` is light-weight and uses a combination of image preprocessing techniques for character extraction and an LSTM-based OCR engine.

(3) Optical Character Recognition: `EasyOCR` (AI, 2020) is used as a backup for `Tesseract` OCR which can fail in more complicated and noisy images. The tools that utilize `EasyOCR` are the same as above. `EasyOCR` uses a VGG16-based `CRAFT` (Character Region Awareness for Text Detection, 138M parameters) for text detection followed by a CRNN (Convolutional Recurrent Neural Network, 83M parameters) model for recognition.

F.2 Tool Outputs and Intermediate Visualizations for Self-Verification in `ChartAgent`

Our chart-specialized tools are carefully designed to produce clear, perception-friendly visualizations and outputs that `ChartAgent` can interpret for self-verification. Figures 8 and 9 show illustrative intermediate visualizations and final outputs from our universal and chart-specific tools, respectively, and also highlight the variations that these tools are able to robustly handle. To support explicit visual inspection, tool outputs include overlays, highlights, or annotations that are optimized to be easily interpretable by the base MLLM (e.g., colored segment overlays in pie charts, bar height markers, annotated legends). These custom-designed artifacts allow `ChartAgent` to reason over visual evidence grounded in the charts. When outputs appear semantically inconsistent or visually incorrect (e.g., pie segments too small, mismatched colors, negative bar heights, or responses contradicting axis values), `ChartAgent` engages in a recover-and-retry process—tweaking tool parameters or invoking alternative tools. This iterative correction loop mimics human-like debugging, ensuring robust reasoning and accurate interpretation in the chart domain. These visualizations are therefore critical for enabling `ChartAgent` to assess intermediate results and adapt its behavior in subsequent steps. A quantitative evaluation of the effectiveness of this visual self-verification is provided in Section 5.3.

Note that some tools generate additional outputs not displayed here—for example, the `annotate_legend` tool also produces a cropped legend image, an annotated cropped legend image, and a bounding-box mapping between detected markers/text and their (x, y, w, h) coordinates. In this figure, however, we highlight only the key output (the annotated cropped legend image) to focus on the most relevant artifacts for visual self-verification. In contrast, some tools produce

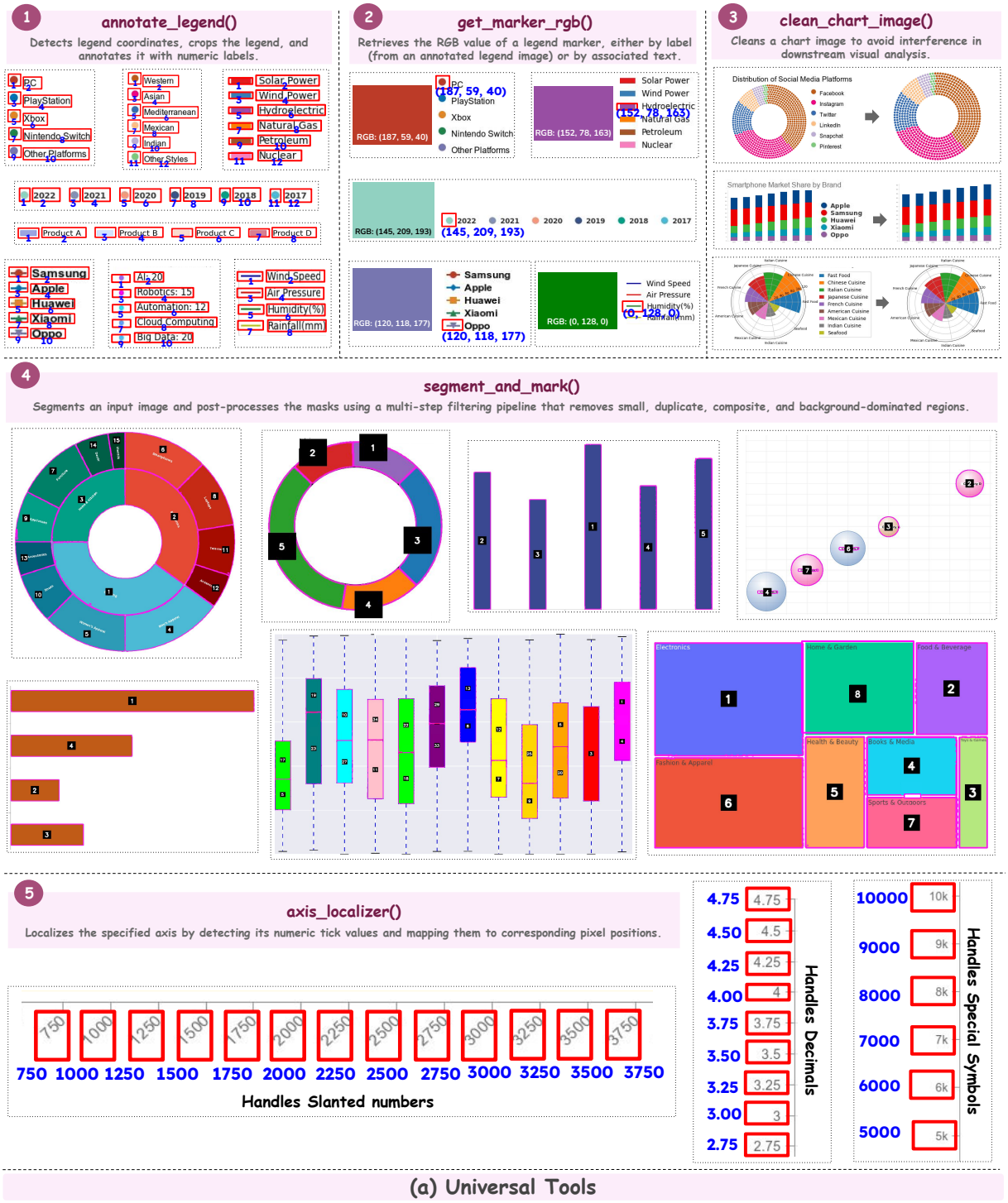


Figure 8: **Illustrative examples of key intermediate and final output visualizations for universal tools in `ChartAgent`.** These visualizations are critical to facilitating visual self-verification in `ChartAgent`. Such tool observations enable `ChartAgent` to perceptually assess the outputs and refine its tool usage in the next iteration—either by adjusting tool parameters or invoking a different tool if the intermediate results indicate incorrect or unexpected behavior. Note the diverse variations that our tools are capable of handling robustly.

only numeric outputs, such as `arithmetic` and `interpolate_pixel_to_value`, which are not included here. Complete input–output specifications for each chart-specialized tool are provided in Table 5 and Section M.1.2.

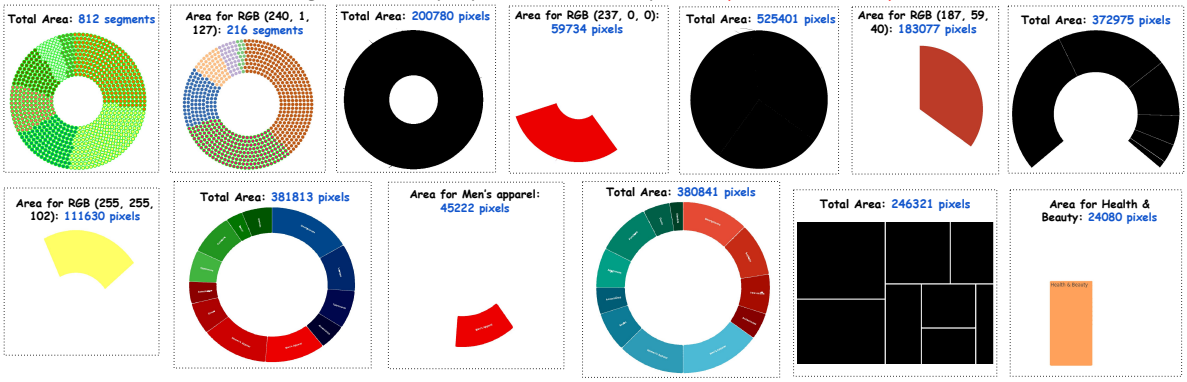
G Implementation Details

G.1 `ChartAgent`

`ChartAgent` is built using the AutoGen 0.2.26 framework, running on Python 3.9 and con-

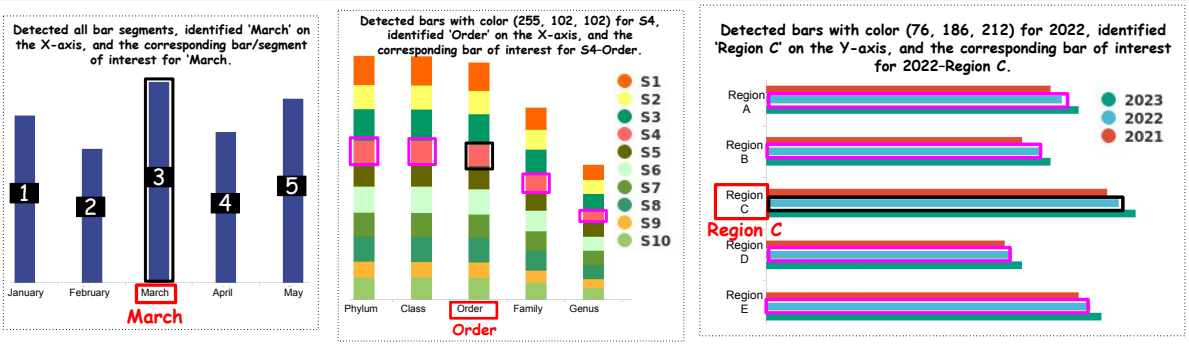
1 compute_segment_area()

Computes the area of a chart segment by: (1) counting discrete visual elements of a specified color, (2) counting pixels of a specified color, or (3) counting pixels within a segment identified by a specific label ID. Commonly used for pie charts and tree maps.



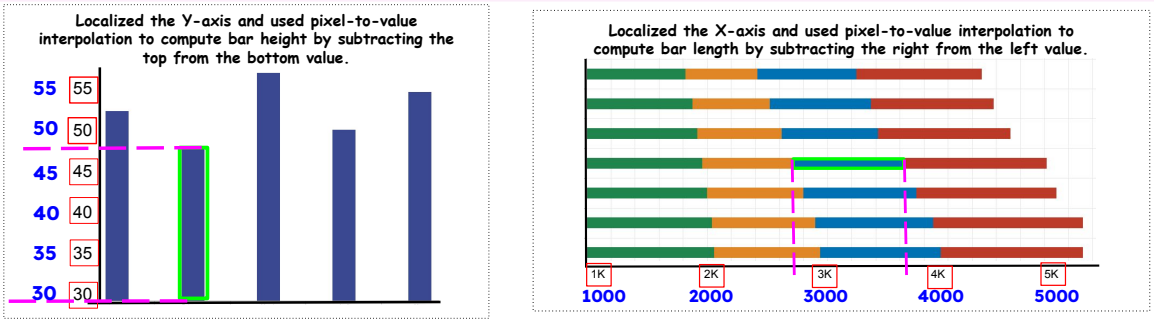
2 get_bar()

Detects and returns the bounding box of a bar in a chart image that matches a specified color and/or axis label. It segments bar regions using a model, filters by color if provided, locates the target axis label using OCR if specified, and selects the closest matching bar accordingly. Commonly used for bar charts.



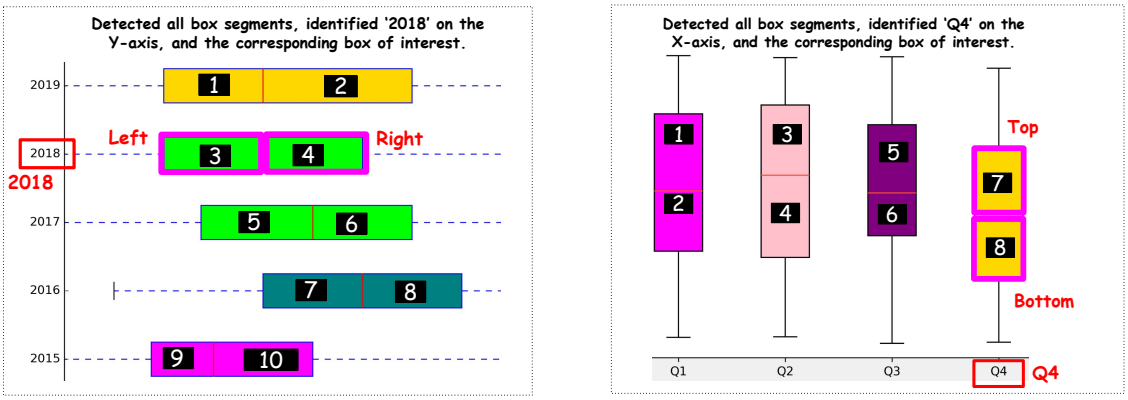
3 compute_bar_height()

Computes a bar's value (height or length) by mapping its pixel bounding box to axis values using OCR-based axis localization. Supports left/right y-axes for vertical bars and the x-axis for horizontal bars. Commonly used for bar charts.



4 get_boxplot()

Detects and returns box plot segments filtered by color, axis label, or segmentation indices. Commonly used for box plots.



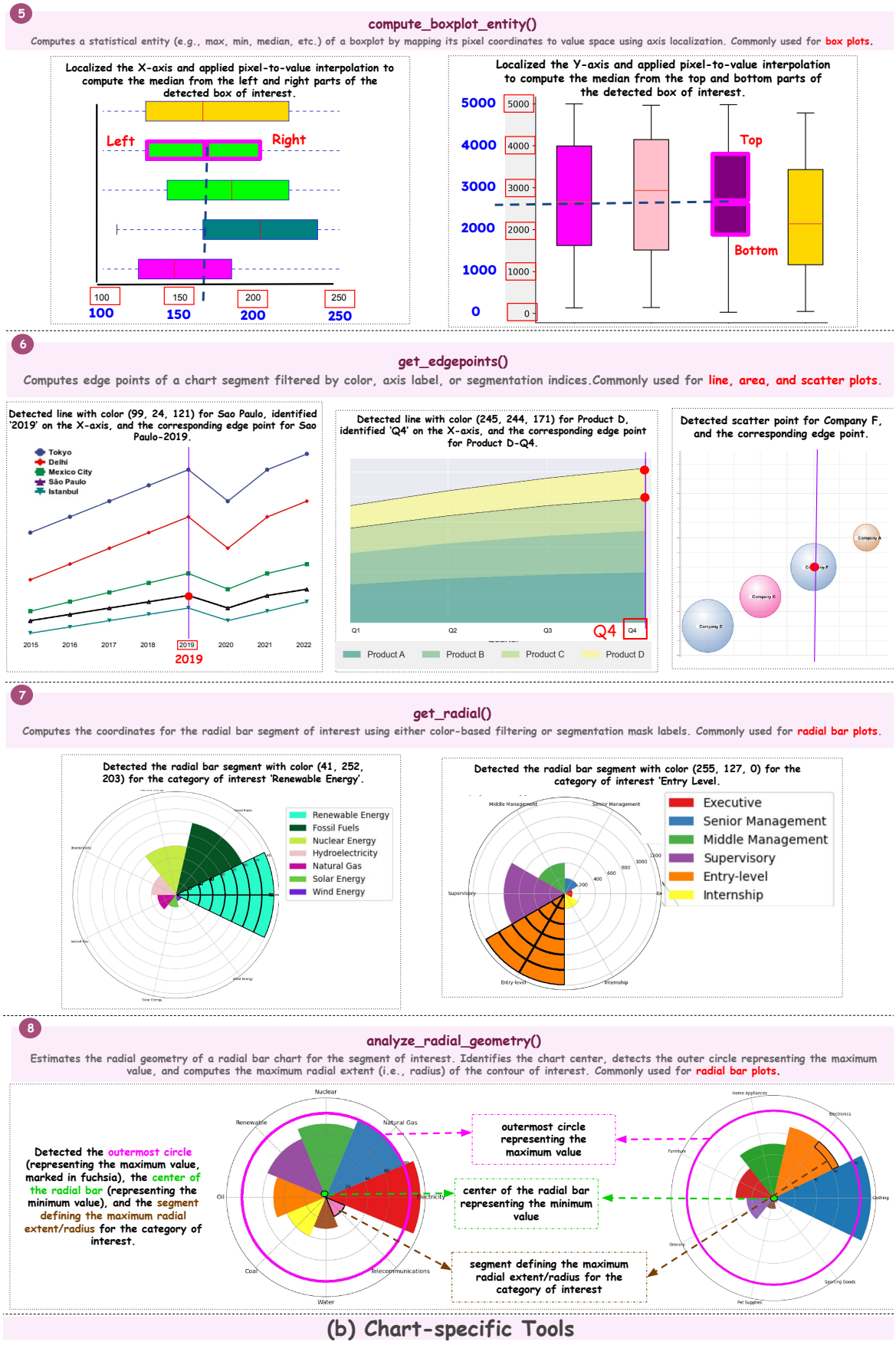


Figure 9: **Illustrative examples of key intermediate and final output visualizations for chart-specific tools in `ChartAgent`.** These visualizations enable visual self-verification in `ChartAgent`, allowing it to refine tool usage through perceptual assessment and iterative correction. We intentionally present some easier examples here for illustration, to help readers quickly follow the process. However, `ChartAgent` tools are capable of handling a wide range of cases, including more difficult and complex ones, as demonstrated by the overall results.

figured to perform a maximum of 15 reasoning iterations per task. The GPT-4o model (gpt-4o-2024-08-06) is used as the base multimodal LLM for reasoning in ChartAgent, with the temperature set to 0.0 to ensure deterministic outputs. Further, for variants of ChartAgent with different base MLLMs, we use GPT-4o-mini (gpt-4o-mini-2024-07-18), Claude 3 Haiku (claude-3-haiku-20240307), and Pixtral-12B-2409. For reproducibility, the random seed is set to 42. All experiments are conducted on a Linux machine using an AWS g4dn.xlarge instance, which is equipped with a single NVIDIA T4 GPU with 16 GB of memory. For segmentation tasks, we use the Segment Anything (ViT-H) (Kirillov et al., 2023) model as the default, which has a size of 2.56 GB and 641 million parameters. For OCR tasks, we utilize Tesseract OCR (Smith, 2007) and EasyOCR (AI, 2020). This setup facilitates efficient execution of both model inference and vision tool pipelines. All ChartAgent prompts are provided in Appendix M.1, and our codebase https://anonymous.4open.science/r/ChartAgent.

G.2 Baselines

Similar to the ChartAgent setup, all applicable baselines were run with a temperature setting of 0.0 to ensure deterministic outputs, with the random seed fixed at 42 for reproducibility. All proprietary baseline models, as well as open-weight general-purpose baseline models, were evaluated using both zero-shot and Chain-of-Thought (CoT) prompting styles. All baseline prompts are provided in Appendix M.2. For chart-based baseline models such as DePlot (Liu et al., 2023a) and OneChart (Chen et al., 2024), which output structured tables rather than direct answers, we apply a zero-shot GPT-4o call to extract the final answer (see Appendix M.2.4 for the corresponding prompt).

H Examples of Response Standardization for Accuracy Evaluation

As part of our two-step accuracy evaluation (Section 4), we use GPT-4o to standardize both the model's response and the ground truth answer, before applying an arithmetic or string-matching correctness check. Below are representative examples of the standardization operations applied:

(1) Converting Scales e.g., K for thousand, M for million, B for billion

- ground truth: 3000 | response: 4K → 4000
- ground truth: 15% → 15 | response: 0.15 times → 15% → 15
- ground truth: 2000m → 2000 | response: 2.5km → 2500m → 2500
- ground truth: 48 hours → 48 | response: 2 days → 48 hours → 48

(2) Stripping Units e.g., $, %, K, M, B, etc.

- ground truth: 5 | response: 5K → 5
- ground truth: 15 | response: 10% → 10

(3) Removing Symbols

- response: 1,000 → 1000

(4) Standardizing Number Formats

- ground truth: 7 | response: seven → 7

These standardizations of the ground truth and response ensure that formatting differences do not lead to incorrect evaluations during the subsequent arithmetic correctness check or string-matching step. Prompts for both evaluation strategies—namely, our standardization-based accuracy computation and the LLM-as-a-Judge baseline evaluation—are provided in Appendix M.3.

I Complexity Analysis

To examine ChartAgent performance under varying levels of difficulty, we divide all chart–QA samples across our evaluation datasets into difficulty levels based on (a) the *visual complexity* of charts and (b) the *reasoning complexity* of chart–QA pairs. This stratification enables us to analyze performance trends across distinct categories of challenge. Each dimension is categorized into three levels: Easy, Medium, and Hard.

- *Visual complexity* reflects the effort needed to interpret the chart image. Easy charts (e.g., single bar or line plots) contain few elements and clean layouts. Medium charts (e.g., multi-series line plots, grouped/stacked bar charts) introduce moderate clutter and overlapping elements. Hard charts (e.g., radar charts, 3D plots, or heavily layered visuals) are highly cluttered and visually demanding.

Table 6: **Complexity Label Statistics.** Distribution of difficulty levels stratified by (a) *visual complexity* of charts and (b) *reasoning complexity* of chart–QA pairs in the evaluation datasets. Rows correspond to reasoning complexity; columns correspond to visual complexity. Each dimension has three levels: Easy , Medium , Hard .

Reasoning Complexity	Visual Complexity			Total
	Easy	Medium	Hard	
Easy	37.38%	35.88%	1.43%	74.69%
Medium	0.76%	8.86%	6.40%	16.02%
Hard	0.98%	7.07%	1.24%	9.29%
Total	39.12%	51.81%	9.07%	100%

(a) ChartBench Dataset

Reasoning Complexity	Visual Complexity			Total
	Easy	Medium	Hard	
Easy	44.27%	20.83%	2.60%	67.71%
Medium	9.38%	7.55%	5.90%	22.74%
Hard	0.52%	3.12%	5.82%	9.55%
Total	54.17%	31.51%	14.32%	100%

(b) ChartX Dataset

(a) Visual Complexity **(b) Reasoning Complexity**

Figure 10: **Complexity dimensions in chart–QA pairs.** Representative examples are shown for (a) *visual complexity* of charts and (b) *reasoning complexity* of chart–QA pairs, each categorized into Easy, Medium, and Hard levels. **(a)** For *visual complexity*: Easy charts (e.g., single bar or line plots) have few elements and clean layouts; Medium charts (e.g., multi-series line or stacked bar plots) add moderate overlap; Hard charts (e.g., radar charts, 3D plots, or heavily layered visuals) are highly cluttered. **(b)** For *reasoning complexity*: Easy chart–QA pairs involve direct lookup; Medium pairs require comparisons or proportions; Hard pairs need complex multi-step reasoning.

- *Reasoning complexity* captures the cognitive effort required to answer a question using the chart. Easy chart–QA pairs involve direct value lookup. Medium pairs require comparisons, ratios, or proportions. Hard pairs demand multi-step reasoning, arithmetic aggregation, or complex logical inference.

Table 6 reports the distribution of visual and reasoning complexity across our evaluation datasets, ChartBench and ChartX. Both datasets provide coverage across all three categories. The majority of charts fall under visually Easy or Medium categories, with fewer than 15% classified as visually Hard. ChartX contains a larger fraction of visually Hard charts, making it slightly more challenging overall in terms of clutter and layout. A similar trend is observed for reasoning complexity: although Easy dominates, both datasets include substantial portions of Medium and Hard reasoning tasks, ensuring coverage of non-trivial scenarios.

Further, Figure 10 illustrates representative examples spanning different chart types and subtypes across the Easy, Medium, and Hard levels for both visual and reasoning complexity. The prompts used to label chart images and chart–QA pairs into these stratified levels are provided in Appendix M.4.

J Qualitative Analysis

J.1 Illustrative Agent Trajectories

J.1.1 Agent Trajectories on Unannotated Charts

Figures 11-17 show sample reasoning trajectories for `ChartAgent` on questions involving diverse unannotated chart types.[1] The LLM-based orchestrator agent classifies the chart as unannotated, triggering the ReAct routine with chart tools. It also retrieves few-shot ICL examples specific to the corresponding chart type, after which the multi-turn interaction loop produces the accurate final answer.

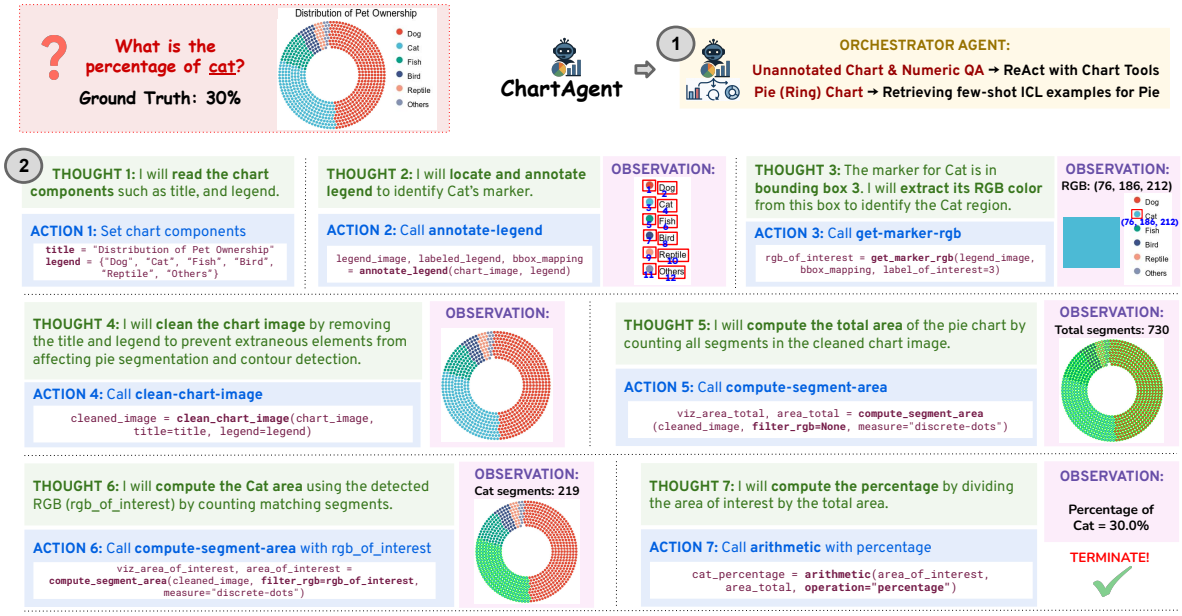


Figure 11: **Qualitative Trajectory on a Pie (Ring) Chart.**

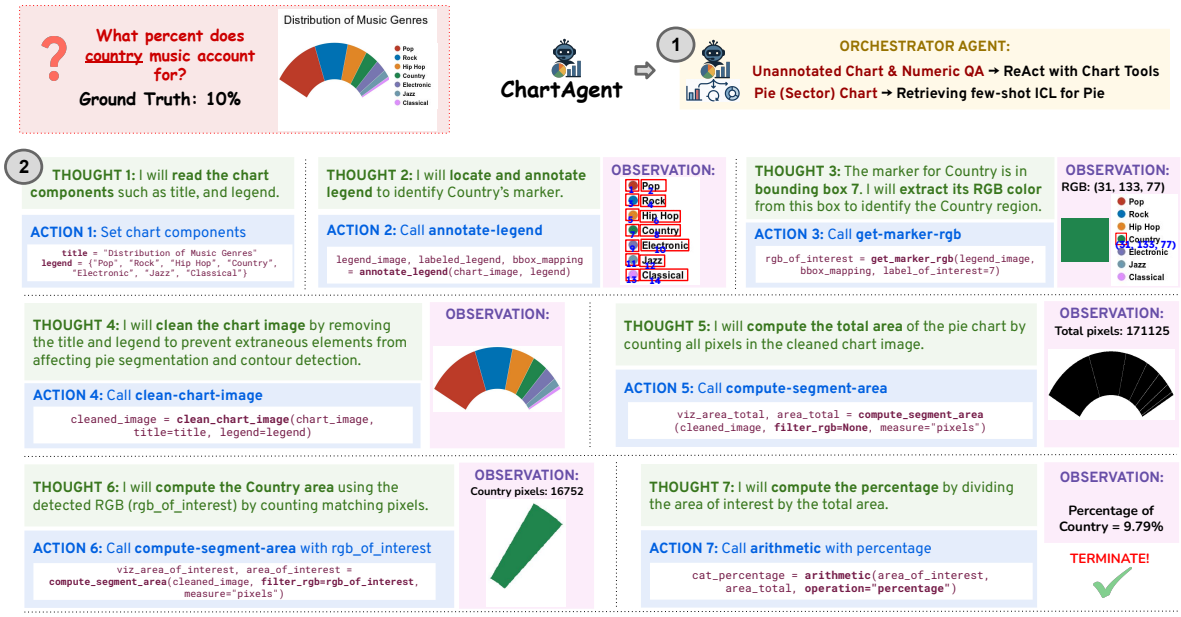


Figure 12: **Qualitative Trajectory on a Pie (Sector) Chart.**

[1] Note: We show examples that highlight a diverse range of chart types rather than variations in reasoning complexity, as our goal here is to focus on the visual reasoning steps in the trajectory examples. However, our datasets include charts spanning all levels of both visual and reasoning difficulty.

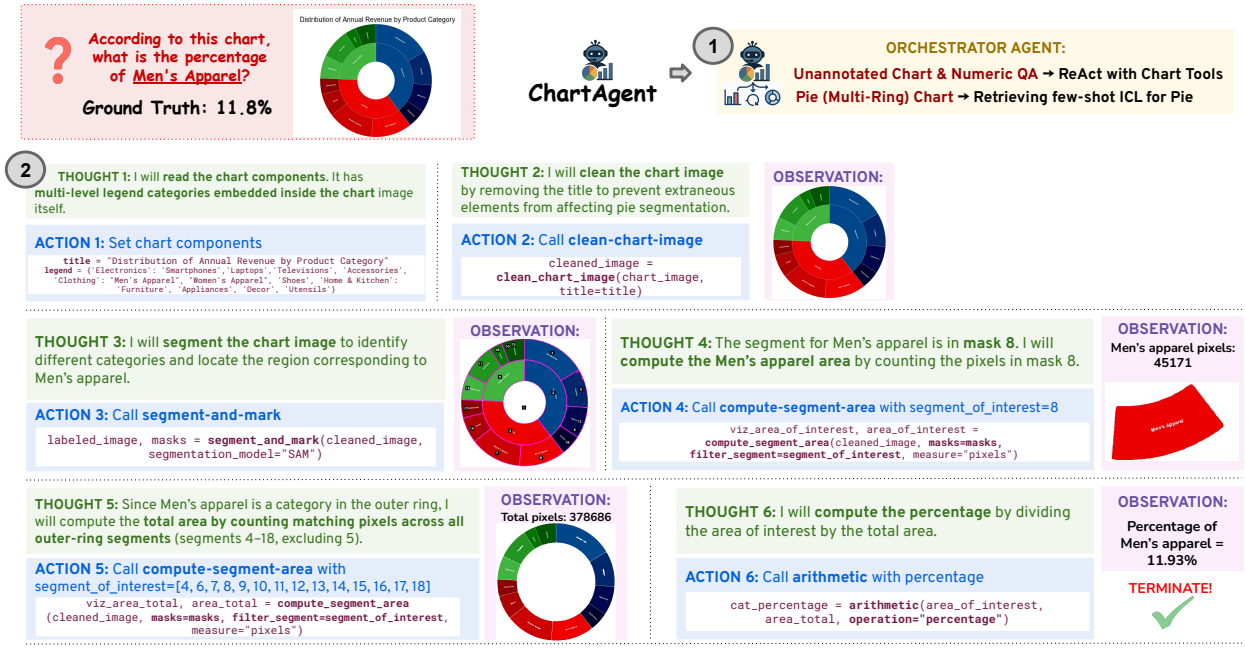


Figure 13: **Qualitative Trajectory on a Pie (Multi-Ring) Chart.**

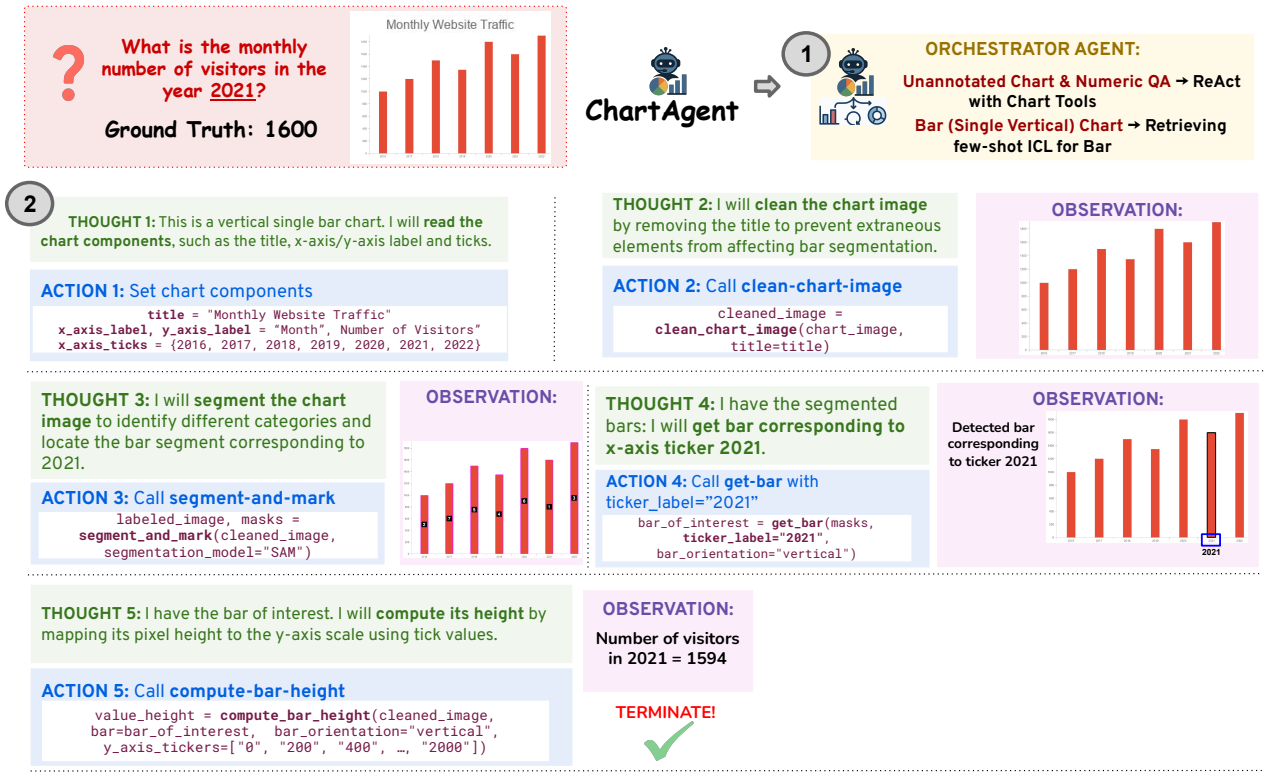


Figure 14: **Qualitative Trajectory on a Bar (Single Vertical) Chart.**

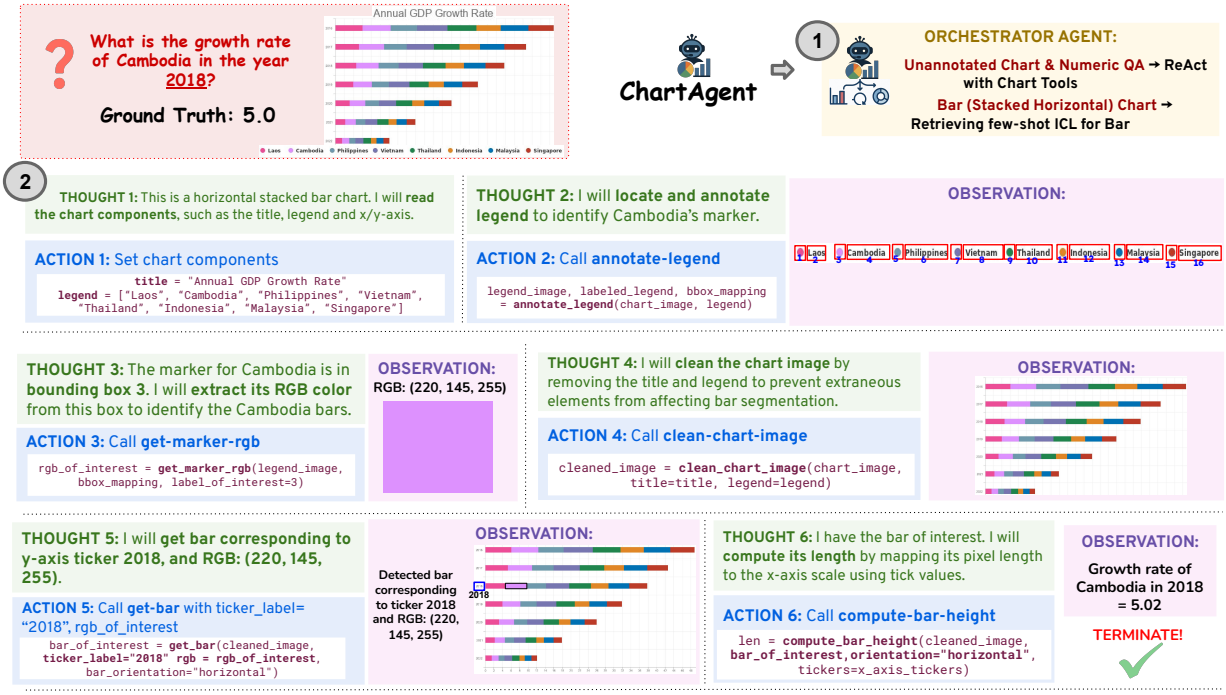


Figure 15: **Qualitative Trajectory on a Bar (Stacked Horizontal) Chart.**

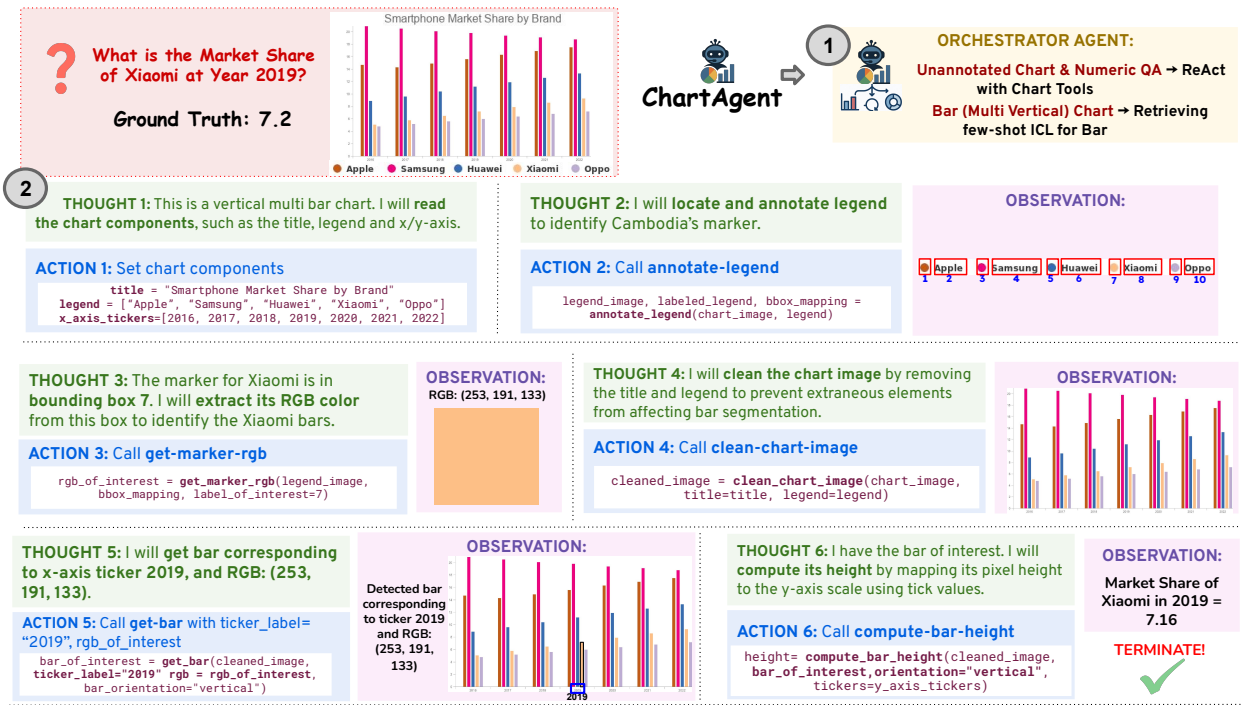


Figure 16: **Qualitative Trajectory on a Bar (Multi-grouped Vertical) Chart.**

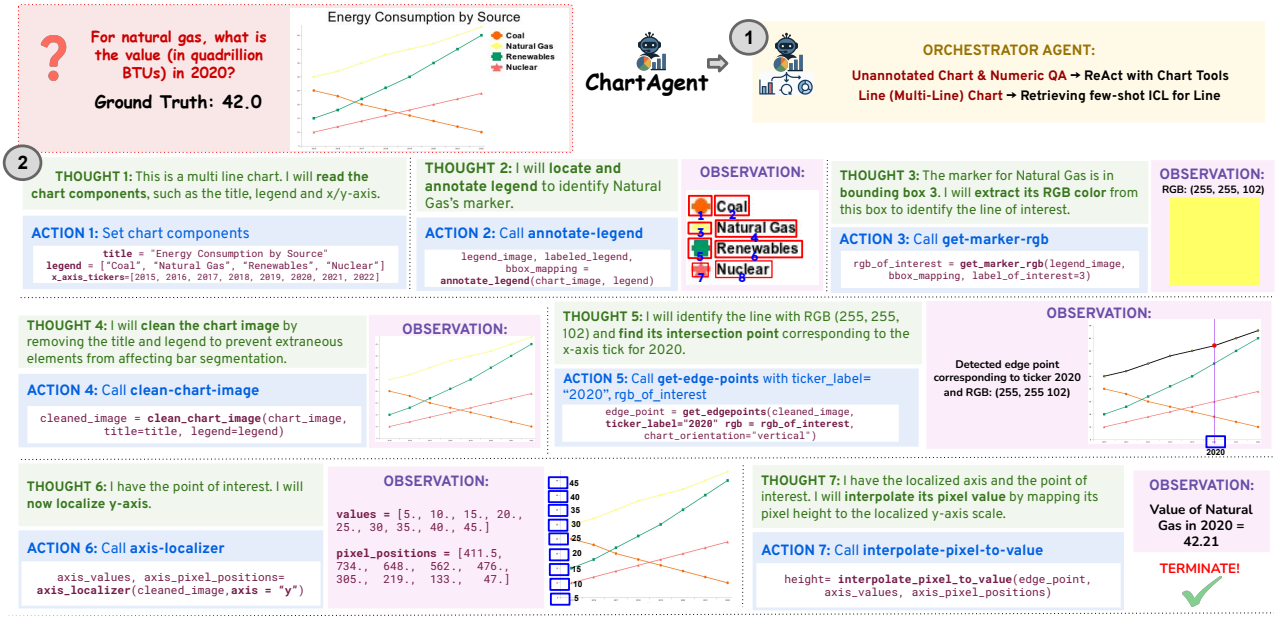


Figure 17: **Qualitative Trajectory on a Line (Multi-line) Chart.**

J.1.2 Agent Trajectories on Annotated Charts

Figure 18 shows sample reasoning trajectories for `ChartAgent` on questions involving annotated charts. The LLM-based orchestrator classifies the chart as annotated and routes it to direct MLLM reasoning (GPT-4o), which produces the correct answer.

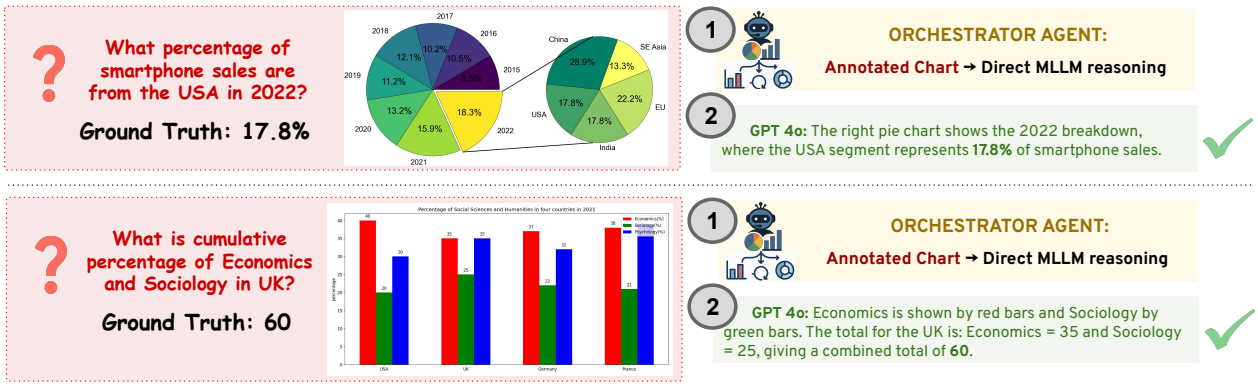


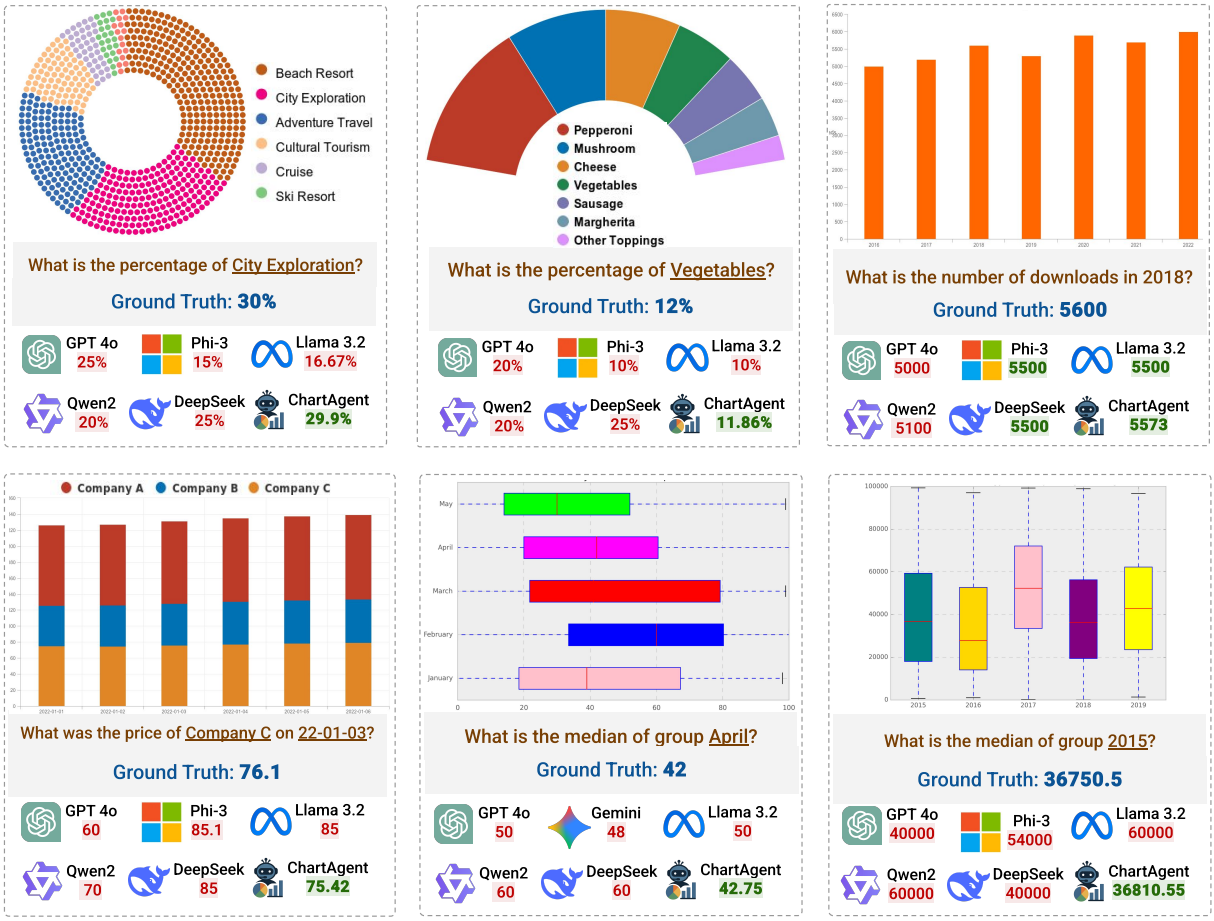
Figure 18: **Qualitative Trajectories on Annotated Chart Examples.**

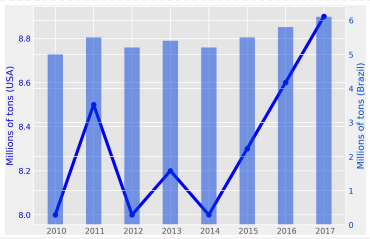
J.1.3 Some Interesting Agent Trajectories

`ChartAgent` exhibits adaptive decision-making during reasoning. For instance, in scatter plots with variable-sized points, it correctly identifies when certain points are too small to be captured through segmentation and instead relies on its own visual judgment to infer the answer—yielding accurate results without tool assistance. Similarly, when tool-based methods fail, the agent provides transparent and reasonable justifications for reverting to direct reasoning. For example: "*THOUGHT 6: The interpolation failed because there is only one y-axis value available. I will directly estimate the Click-through Rate from the chart image using the visual position of the Campaign F bubble. ANSWER: The Click-through Rate for Campaign F when the Impressions is 700 is approximately 5.5%. TERMINATE.*" Such cases highlight `ChartAgent`'s ability to recognize tool limitations and intelligently switch to self-guided reasoning when appropriate.

J.2 Representative Examples

Figure 19 presents qualitative comparison examples across the diverse chart types that `ChartAgent` can handle, alongside several current SoTA methods. We observe improved performance across the variety of chart types in both the ChartBench and ChartX datasets.



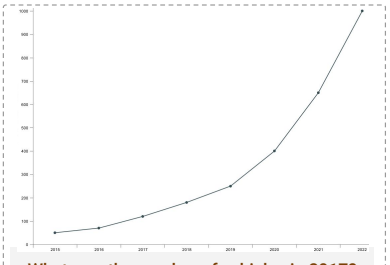


What was Brazil's consumption in 2011?

Ground Truth: 5.5

GPT 4o 4.0 | Phi-3 4.3 | Llama 3.2 8.8

Qwen2 8.8 | DeepSeek 8.6 | ChartAgent 5.41

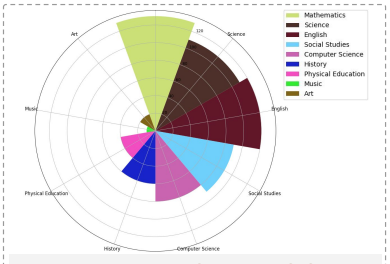


What was the number of vehicles in 2017?

Ground Truth: 120

GPT 4o 150 | Phi-3 251.8 | Llama 3.2 150

Qwen2 118.0 | DeepSeek 150 | ChartAgent 123.4

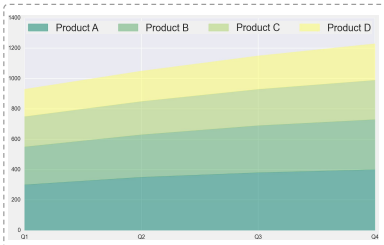


How many more students in Math than in Physical Education?

Ground Truth: 90

GPT 4o 80 | Phi-3 20 | Llama 3.2 100

Qwen2 60 | DeepSeek 40 | ChartAgent 89.8

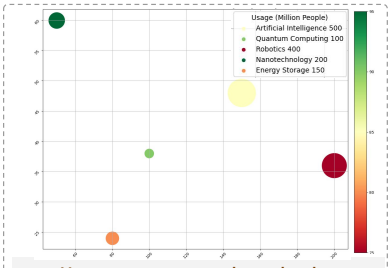


What was the sales of Product D in Q4?

Ground Truth: 240

GPT 4o 200 | Gemini 250 | Llama 3.2 1200

Qwen2 1250 | DeepSeek 1200 | ChartAgent 241.31



How many more months to develop nanotechnology vs. energy storage?

Ground Truth: 36

GPT 4o 15 | DeepSeek 35 | Llama 3.2 120

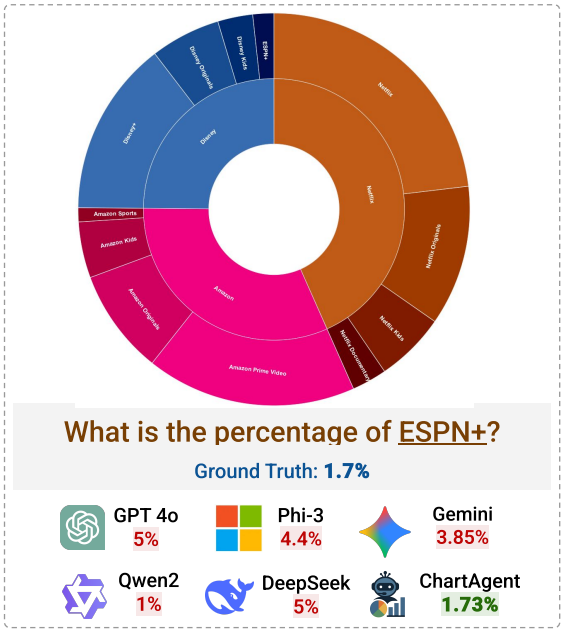
Qwen2 Non-numeric | Phi-3 24 | ChartAgent 35

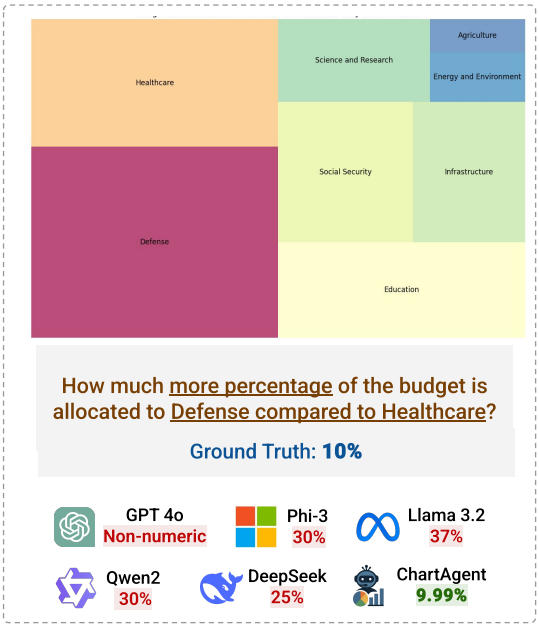
For Company F, what was the volume when the price was 120.0?

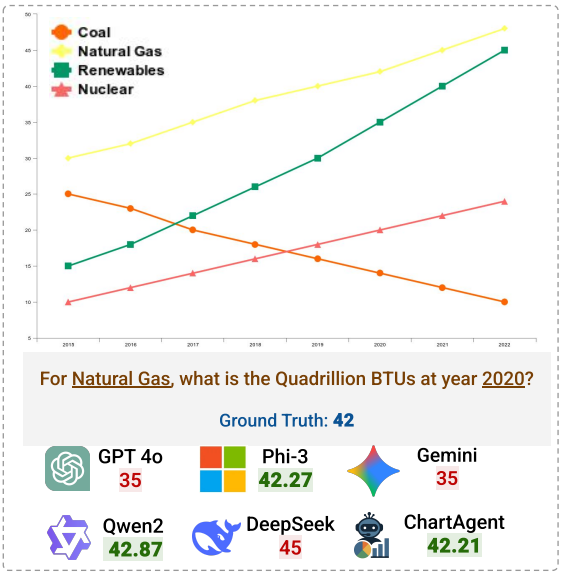
Ground Truth: 600

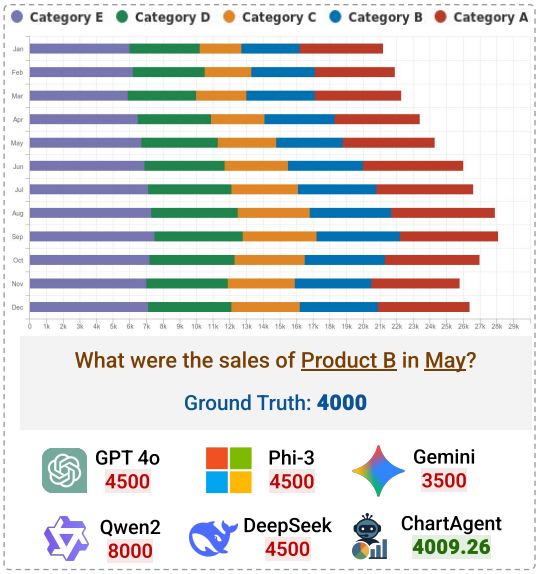
GPT 4o 500 | Phi-3 580 | Llama 3.2 250

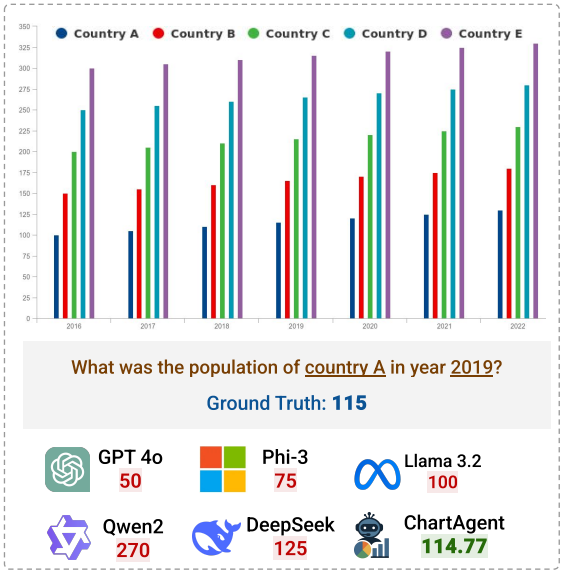
Qwen2 650 | DeepSeek 650 | ChartAgent 600











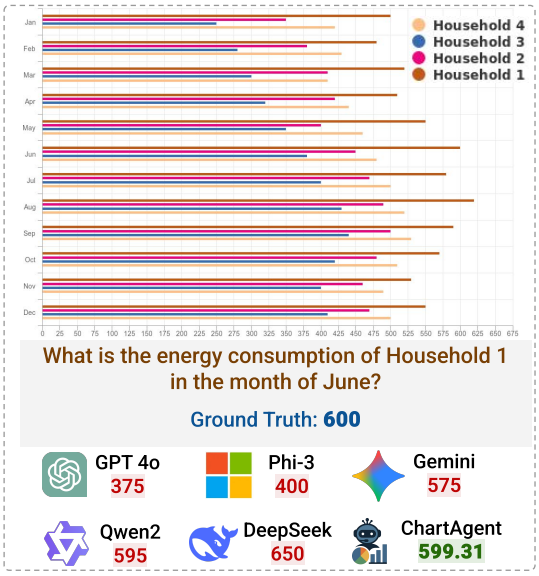


Figure 19: **Qualitative Examples.** Correct responses (within a 5% error margin) are highlighted in green, while incorrect responses are highlighted in red.

K Expanded Discussion on Results

K.1 Performance by Chart Type

Table 7 shows the accuracy on unannotated charts by chart type.

Table 7: **Accuracy on unannotated charts (%) by chart type. Red:** Best, Blue: Second best. **Abbreviations:** Over: Overlay | Stack: Stacked | Mul: Multi | Sing: Single | Hor: Horizontal | Vert: Vertical | B-L: Bar-Line | L-L: Line-Line | Dir: Directed | Undir: Undirected | Combo: Combination. See App. D for examples of each chart type.

Model	Area		Horizontal Bar			3D Bar		Vertical Bar			Box			Combo		Line		Node		Pie			Radar			Scatter	Avg. ↑
	Over	Stack	Mul	Sing	Stack	Mul	Stack	Mul	Sing	Stack	Hor	Vert	Stock	B-L	L-L	Mul	Sing	Dir	Undir	Mul	Ring	Sector	Mul	Fill	Sing	3D	
Proprietary Multimodal Large Language Models																											
GPT 4o	21.0	18.0	24.0	59.0	10.0	20.0	6.0	38.0	73.0	12.0	20.0	26.0	63.0	35.0	41.0	37.0	75.0	**91.0**	**91.0**	3.0	32.0	34.0	22.0	20.0	6.0	63.0	36.15
GPT 4o-mini	23.0	7.0	13.0	27.0	7.0	20.0	7.0	19.0	56.0	2.0	13.0	12.0	57.0	29.0	36.0	19.0	50.0	88.0	**91.0**	1.0	7.0	16.0	3.0	8.0	1.0	43.0	25.19
Claude 3	15.0	5.0	12.0	32.0	7.0	25.0	5.0	51.0	67.0	6.0	8.0	5.0	62.0	24.0	23.0	28.0	50.0	75.0	71.0	7.0	9.0	12.0	3.0	13.0	11.0	51.0	26.04
Gemini 1.5	5.0	4.0	28.0	52.0	7.0	14.0	4.0	39.05	49.0	5.0	13.0	18.0	24.0	28.0	5.0	7.0	**91.0**	48.0	59.26	1.0	14.0	29.52	1.0	7.0	0.0	45.0	27.27
Open-weights Multimodal Large Language Models																											
BLIP-2	0.0	0.0	3.0	1.0	4.0	5.0	4.0	2.0	4.0	3.0	3.0	1.0	3.0	0.0	0.0	4.0	4.0	3.0	5.0	3.0	2.0	2.0	9.0	2.0	6.0	3.0	2.92
CogAgent	14.0	2.0	3.0	15.0	6.0	15.0	4.0	11.0	9.0	4.0	8.0	6.0	22.0	21.0	16.0	6.0	20.0	20.0	31.0	3.0	18.0	9.0	2.0	4.0	13.0	20.0	11.62
CogVLM	21.0	3.0	4.0	17.0	3.0	18.0	3.0	11.0	16.0	4.0	7.0	7.0	2.0	24.0	20.0	9.0	10.0	19.0	24.0	1.0	7.0	25.0	13.0	15.0	6.0	16.0	11.62
DeepSeek-VL2	29.0	11.0	25.0	57.0	8.0	36.0	8.0	58.0	82.0	13.0	11.0	3.0	51.0	46.0	48.0	51.0	8.0	31.0	36.0	0.0	6.0	15.0	13.0	21.0	5.0	44.0	30.31
DocOwl1.5	19.0	8.0	21.0	69.0	3.0	20.0	0.0	39.0	78.0	6.0	7.0	17.0	32.0	15.0	23.0	23.0	74.0	42.0	47.0	2.0	14.0	8.0	2.0	14.0	10.0	20.0	23.58
InstructBLIP	5.0	7.0	3.0	11.0	1.0	5.0	4.0	3.0	11.0	4.0	4.0	1.0	1.0	3.0	5.0	2.0	9.0	23.0	26.0	2.0	1.0	3.0	2.0	7.0	0.0	11.0	5.92
InternVL3	25.0	16.0	45.0	80.0	19.0	38.0	1.0	44.0	80.0	16.0	16.0	23.0	60.0	27.0	24.0	30.0	56.0	62.0	52.0	0.0	2.0	9.0	**24.0**	24.0	6.0	25.0	30.92
LLama3.2	46.0	21.0	58.0	91.0	11.0	31.0	4.0	71.0	89.0	10.0	6.0	6.0	49.0	42.0	46.0	63.0	87.0	42.0	58.0	5.0	4.0	25.0	8.0	17.0	10.0	46.0	36.38
Llava1.6	7.0	7.0	11.0	12.0	8.0	18.0	1.0	7.0	19.0	1.0	5.0	3.0	0.0	16.0	15.0	7.0	5.0	39.0	45.0	1.0	4.0	5.0	3.0	1.0	2.0	16.0	9.92
Llava1.5	1.0	5.0	8.0	12.0	7.0	6.0	3.0	5.0	9.0	4.0	4.0	1.0	2.0	7.0	1.0	3.0	5.0	11.0	22.0	0.0	8.0	11.0	9.0	13.0	11.0	14.0	7.00
LlaVA-OneVision	9.0	2.0	9.0	7.0	12.0	12.0	10.0	11.0	7.0	7.0	12.0	8.0	14.0	7.0	10.0	2.0	5.0	38.0	36.0	0.0	1.0	1.0	24.0	12.0	1.0	16.0	10.50
mPLUG-Owl3	11.0	2.0	9.0	20.0	1.0	15.0	2.0	11.0	15.0	2.0	7.0	6.0	16.0	14.0	15.0	14.0	10.0	52.0	41.0	0.0	10.0	23.0	7.0	17.0	3.0	6.0	12.65
Phi3-vision	27.0	37.0	43.0	78.0	8.0	**40.0**	7.0	86.0	92.0	30.0	9.0	15.0	48.0	31.0	55.0	66.0	84.0	39.0	51.0	2.0	14.0	21.0	11.0	**26.0**	**66.0**	**73.0**	40.77
Pixtral	26.0	10.0	25.0	51.0	6.0	30.0	5.0	39.0	89.0	10.0	16.0	29.0	39.0	19.0	24.0	17.0	32.0	68.0	59.0	2.0	21.0	28.0	13.0	9.0	8.0	72.0	28.73
Qwen2VL	**57.0**	18.0	**87.0**	**97.0**	17.0	**40.0**	7.0	**94.0**	**97.0**	24.0	13.0	4.0	**64.0**	37.0	46.0	**80.0**	85.0	80.0	86.0	1.0	12.0	9.0	9.0	11.0	9.0	47.0	43.50
QwenVLChat	6.0	8.0	4.0	8.0	2.0	6.0	3.0	5.0	17.0	5.0	0.0	1.0	2.0	9.0	7.0	6.0	6.0	20.0	22.0	2.0	2.0	3.0	8.0	3.0	10.0	5.0	6.54
SmolVLM	7.0	3.0	12.0	17.0	3.0	12.0	1.0	14.0	26.0	0.0	7.0	7.0	28.0	15.0	13.0	5.0	23.0	62.0	54.0	0.0	2.0	12.0	14.0	16.0	9.0	14.0	14.46
SPHINX-V	7.0	2.0	3.0	17.0	4.0	16.0	10.0	9.0	26.0	4.0	4.0	7.0	2.0	16.0	22.0	7.0	10.0	46.0	54.0	2.0	3.0	16.0	4.0	8.0	14.0	7.0	12.30
VisualGLM	6.0	3.0	1.0	2.0	4.0	2.0	1.0	4.0	6.0	5.0	1.0	6.0	0.0	0.0	2.0	6.0	3.0	63.0	53.0	1.0	5.0	4.0	7.0	4.0	2.0	8.0	7.65
Chart-related Models																											
ChartGemma	25.0	8.0	21.0	54.0	9.0	21.0	3.0	36.0	86.0	6.0	5.0	5.0	22.0	31.0	36.0	24.0	68.0	32.0	38.0	0.0	2.0	8.0	3.0	8.0	3.0	29.0	22.42
ChartInstruct	20.0	6.0	23.0	72.0	1.0	17.0	7.0	36.0	85.0	6.0	9.0	27.0	5.0	27.0	24.0	13.0	68.0	18.0	26.0	2.0	8.0	3.0	8.0	6.0	4.0	4.0	20.19
ChartLlama	20.0	2.0	2.0	15.0	7.0	12.0	7.0	14.0	20.0	7.0	5.0	9.0	1.0	16.0	18.0	3.0	10.0	41.0	38.0	2.0	8.0	15.0	0.0	0.0	11.0	14.0	11.42
ChartVLM	16.0	8.0	24.0	78.0	10.0	29.0	7.0	60.0	85.0	8.0	3.0	23.0	7.0	37.0	40.0	30.0	95.0	13.0	10.0	1.0	7.0	5.0	2.0	4.0	6.0	14.0	23.92
DePlot	18.0	2.0	43.0	74.0	13.0	34.0	9.0	66.0	78.0	7.0	20.0	20.0	0.0	48.0	45.0	14.0	63.0	84.0	73.0	4.0	3.0	5.0	2.0	2.0	3.0	2.0	28.15
MatCha	3.0	1.0	8.0	29.0	0.0	8.0	1.0	18.0	40.0	11.0	3.0	17.0	1.0	16.0	14.0	13.0	18.0	16.0	19.0	0.0	1.0	1.0	2.0	0.0	2.0	10.0	9.69
OneChart	0.0	6.0	27.0	67.0	2.0	16.0	2.0	69.0	80.0	11.0	0.0	17.0	0.0	12.0	62.0	38.0	90.0	65.0	60.0	0.0	0.0	7.0	0.0	0.0	0.0	2.0	26.81
TinyChart	32.0	22.0	71.0	88.0	13.0	37.0	**15.0**	76.0	82.0	21.0	2.0	3.0	4.0	46.0	50.0	51.0	**91.0**	22.0	35.0	1.0	20.0	21.0	10.0	8.0	4.0	27.0	32.77
UniChart	15.0	5.0	24.0	59.0	7.0	11.0	0.0	32.0	60.0	1.0	3.0	8.0	6.0	16.0	25.0	13.0	37.0	36.0	33.0	3.0	0.0	1.0	4.0	4.0	1.0	11.0	15.96
Multimodal Agentic Framework (Ours)																											
ChartAgent	30.0	**38.0**	79.0	76.0	**82.0**	20.0	6.0	88.0	88.0	**76.0**	**89.0**	**83.0**	**64.0**	**67.0**	**65.0**	63.0	81.0	**91.0**	**91.0**	**18.0**	**94.0**	**80.0**	22.0	20.0	6.0	64.0	**60.81**

(a) ChartBench Dataset (9 major chart types, 42 subtypes; 26 unannotated)

Model	Area	Bar	3D Bar	Box	Bubble	Candlestick	Heatmap	Histogram	Line	Multi-Axes	Radar	Ring	Rose	Treemap	Average ↑
Proprietary Multimodal Large Language Models															
GPT 4o	26.0	35.19	22.0	40.0	44.0	**78.0**	50.0	42.55	53.92	18.0	30.0	30.0	**34.0**	44.83	39.44
GPT 4o-mini	16.0	32.41	34.0	42.0	48.0	66.0	50.0	34.04	39.22	8.0	28.0	35.0	26.0	24.14	33.94
Claude 3 Haiku	26.0	25.0	20.0	22.0	38.0	48.0	50.0	27.66	33.33	6.0	22.0	15.0	20.0	10.34	25.77
Gemini 1.5	26.0	40.74	22.0	**48.0**	50.0	8.0	25.0	44.68	33.33	18.0	20.0	30.0	30.0	20.69	31.41
Open-weights Multimodal Large Language Models															
BLIP-2	0.0	0.9	2.0	0.0	2.0	2.0	0.0	2.1	2.0	0.0	6.0	0.0	4.0	0.0	1.69
CogAgent	16.0	23.15	30.0	30.0	20.0	48.0	50.0	19.15	30.39	10.0	26.0	15.0	24.0	17.24	24.93
CogVLM	20.0	31.48	30.0	28.0	16.0	34.0	50.0	17.02	25.49	12.0	26.0	15.0	16.0	27.59	24.23
DeepSeek-VL2	24.0	41.7	24.0	36.0	34.0	62.0	50.0	38.3	54.9	14.0	26.0	20.0	26.0	17.2	35.63
DocOwl1.5	14.0	24.07	20.0	32.0	18.0	44.0	50.0	42.55	35.29	12.0	24.0	5.0	10.0	3.45	24.37
InstructBLIP	6.0	3.7	20.0	14.0	10.0	0.0	25.0	2.1	17.6	8.0	8.0	0.0	6.0	10.3	8.87
InternVL3	24.0	36.11	30.0	44.0	38.0	66.0	50.0	53.19	49.02	16.0	24.0	30.0	32.0	3.45	36.62
LLama3.2	40.0	37.0	30.0	30.0	26.0	58.0	25.0	70.2	69.6	16.0	26.0	25.0	28.0	20.7	39.86
Llava1.6	16.0	19.4	24.0	26.0	12.0	30.0	50.0	14.9	25.5	4.0	18.0	10.0	10.0	3.4	18.17
Llava1.5	12.0	11.1	18.0	36.0	16.0	6.0	0.0	8.5	20.6	8.0	20.0	5.0	10.0	6.9	14.51
LlaVA-OneVision	8.0	12.0	12.0	16.0	10.0	36.0	0.0	6.4	20.6	6.0	8.0	10.0	8.0	0.0	12.82
mPLUG-Owl3	14.0	30.6	24.0	24.0	12.0	18.0	25.0	19.1	22.5	4.0	16.0	5.0	8.0	10.3	18.31
Phi3-vision	38.0	41.7	38.0	54.0	40.0	58.0	50.0	46.8	52.0	22.0	40.0	35.0	36.0	13.8	41.69
Pixtral	34.0	45.4	22.0	54.0	42.0	62.0	50.0	44.7	43.1	14.0	32.0	20.0	24.0	31.0	38.17
Qwen2VL	28.0	**53.70**	**38.0**	42.0	42.0	60.0	50.0	**65.96**	61.76	18.0	26.0	15.0	**34.0**	13.79	42.96
QwenVLChat	24.0	17.59	18.0	20.0	20.0	28.0	50.0	21.28	28.43	6.0	**36.0**	10.0	6.0	13.79	20.42
SmolVLM	26.0	23.15	20.0	28.0	14.0	50.0	0.0	17.02	31.37	8.0	20.0	5.0	16.0	0.0	22.11
SPHINX-V	18.0	20.4	20.0	20.0	16.0	30.0	0.0	21.3	28.4	10.0	30.0	5.0	18.0	13.8	20.70
VisualGLM	16.0	8.33	24.0	10.0	22.0	8.0	**75.0**	8.51	18.63	8.0	16.0	10.0	4.0	6.90	13.10
Chart-related Models															
ChartGemma	**32.0**	36.11	26.0	30.0	28.0	42.0	25.0	31.91	42.16	8.0	22.0	10.0	18.0	6.90	28.87
ChartInstruct	8.0	16.67	12.0	26.0	6.0	56.0	0.0	21.28	28.43	4.0	8.0	5.0	10.0	10.34	17.75
ChartLlama	12.0	18.52	**38.0**	28.0	16.0	44.0	25.0	8.51	24.51	10.0	28.0	15.0	16.0	13.79	21.55
ChartVLM	12.0	26.85	28.0	34.0	26.0	42.0	50.0	42.55	44.12	16.0	24.0	30.0	18.0	13.79	29.01
DePlot	16.0	52.78	14.0	22.0	32.0	32.0	25.0	63.83	70.59	16.0	22.0	5.0	6.0	13.79	34.51
MatCha	12.0	18.5	18.0	12.0	16.0	32.0	50.0	8.5	29.4	6.0	14.0	10.0	10.0	10.3	17.04
OneChart	9.3	69.52	5.26	20.41	10.87	39.58	0.0	63.04	**77.0**	**24.0**	9.3	30.0	11.11	3.57	37.14
TinyChart	22.0	47.22	28.0	28.0	24.0	62.0	25.0	51.06	46.08	16.0	24.0	10.0	16.0	6.90	33.38
UniChart	16.0	23.15	14.0	12.0	4.0	26.0	**75.0**	42.55	29.41	12.0	12.0	10.0	8.0	6.90	18.87
Multimodal Agentic Framework (Ours)															
ChartAgent	**32.0**	50.0	30.0	33.33	**70.0**	50.0	50.0	36.17	64.71	16.0	30.0	**50.0**	28.0	**65.52**	**44.16**

(b) ChartX Dataset (18 chart types in total; 14 unannotated)

K.2 Analysis of Tool Usage in `ChartAgent`

To gain deeper insight into the internal decision-making process of `ChartAgent`, we examine how it selects visual tools across different chart types. Table 8 summarizes the most frequently used tools for each chart type, reflecting tool-usage patterns observed in agent trajectories (see Appendix Table 5 for detailed descriptions of each tool's functionality). This analysis demonstrates that `ChartAgent` strategically adapts its tool usage to the structural and semantic properties of different chart types.

Further, Figure 20 illustrates the percentage of times `ChartAgent` employs each tool across chart types. Overall, **tool usage is strongly chart-type dependent**. Universal tools (e.g., `annotate_legend`, `get_marker_rgb`, `clean_chart_image`) are employed consistently across nearly all chart types, whereas chart-specific tools (e.g., `get_boxplot` for boxplots or `analyze_radial_geometry` for radial bars) are invoked only when structurally required. Combination charts exhibit the highest diversity of tool usage, reflecting the need to simultaneously process multiple chart modalities (e.g., bar and line elements).

Interestingly, several tools show nearly identical usage percentages, suggesting they are frequently used together in agent trajectories. For example, `annotate_legend` and `get_marker_rgb` exhibit very similar distributions across chart types: once the legend is localized, the agent almost always proceeds to extract the corresponding marker color. Such patterns indicate that **certain tools are implicitly coupled in the decision-making process**, with `ChartAgent` invoking them in conjunction to complete semantically linked subtasks.

K.3 Ablation Study

Prior agentic frameworks in natural image VQA rely heavily on generic tools like cropping and zooming. While effective for object localization or text spotting in natural images, these tools lack the capabilities required for structured, quantitative reasoning over charts. Chart-based QA tasks often demand operations such as axis parsing, color-based segmentation, pixel-to-value interpolation, and arithmetic reasoning, which cannot be supported by coarse manipulations like cropping or zooming. This motivates the design of chart-specialized tools tightly integrated into the reasoning loop.

Generic tools such as crop/zoom are insufficient because:

- They cannot extract or match RGB values to identify legend categories.
- They cannot segment visual elements (e.g., pie slices, bars) based on color or structure.
- They cannot compute pixel areas or interpolate numerical values from axes.

As a result, agents using only natural image tools often produce reasoning traces filled with irrelevant observations, ultimately lowering accuracy. In contrast, chart-specialized tools (e.g., axis parsing, bar/pie segmentation, legend detection, numeric estimation) allow precise grounding of reasoning steps and enable recovery via visual self-verification.

To understand the contribution of chart-specialized visual tools in our framework, we conduct an ablation study comparing three variants of the ReAct agent, all implemented with GPT-4o as the underlying reasoning model and equipped with visual self-verification: (i) **ReAct (No Tools)**: reasoning without any visual tools; (ii) **ReAct + Natural Image Tools**: reasoning augmented with generic natural-image tools such as crop and zoom; and (iii) **`ChartAgent` (Ours)**: reasoning supported by chart-specialized tools designed for fine-grained chart understanding.

Table 9 presents the comparison across the three variants. We report both average accuracy and performance on the more challenging subset of unannotated/numeric chart questions.

The results highlight several key observations:

- **ReAct without tools underperforms even GPT-4o + CoT.** While ReAct provides reasoning structure, without visual grounding it accumulates errors, producing misleading traces.
- **Generic tools provide marginal gains.** Crop/zoom adds limited context but cannot handle structured quantitative reasoning, resulting in only minor improvements over no tools.
- **Chart-specialized tools are critical.** The large performance jump with `ChartAgent` demonstrates the necessity of type-specific visual grounding and self-verification mechanisms for robust chart QA.

Table 8: **Most frequently used tools across chart types.** Tool-usage patterns observed in agent trajectories (see Appendix Table 5 for tool descriptions).

Chart Type (Chart Subtypes)	Chart Tools Used
Pie (Ring, Sector, Multi-Ring), Treemap	annotate_legend | get_marker_rgb | clean_chart_image | segment_and_mark | compute_segment_area | arithmetic
Bar (Horizontal/Vertical Single/Multi/Stacked, Histogram, 3D)	annotate_legend | get_marker_rgb | clean_chart_image | segment_and_mark | get_bar | compute_bar_height | axis_localizer | interpolate_pixel_to_value
Box (Horizontal/Vertical)	clean_chart_image | segment_and_mark | get_boxplot | compute_boxplot_entity | axis_localizer | interpolate_pixel_to_value
Area (Overlay, Stacked)	annotate_legend | get_marker_rgb | clean_chart_image | segment_and_mark | get_edgepoints | axis_localizer | interpolate_pixel_to_value | arithmetic
Line (Single/Multi)	annotate_legend | get_marker_rgb | clean_chart_image | get_edgepoints | axis_localizer | interpolate_pixel_to_value
Scatter (Bubble, 3D)	annotate_legend | get_marker_rgb | clean_chart_image | segment_and_mark | get_edgepoints | axis_localizer | interpolate_pixel_to_value
Radial Bar, Rose	annotate_legend | get_marker_rgb | clean_chart_image | segment_and_mark | get_radial | analyse_radial_geometry | estimate_radial_value
Combination (Bar-Line, Line-Line), Multi-Axes	annotate_legend | get_marker_rgb | clean_chart_image | segment_and_mark | get_bar | compute_bar_height | get_edgepoints | axis_localizer | interpolate_pixel_to_value

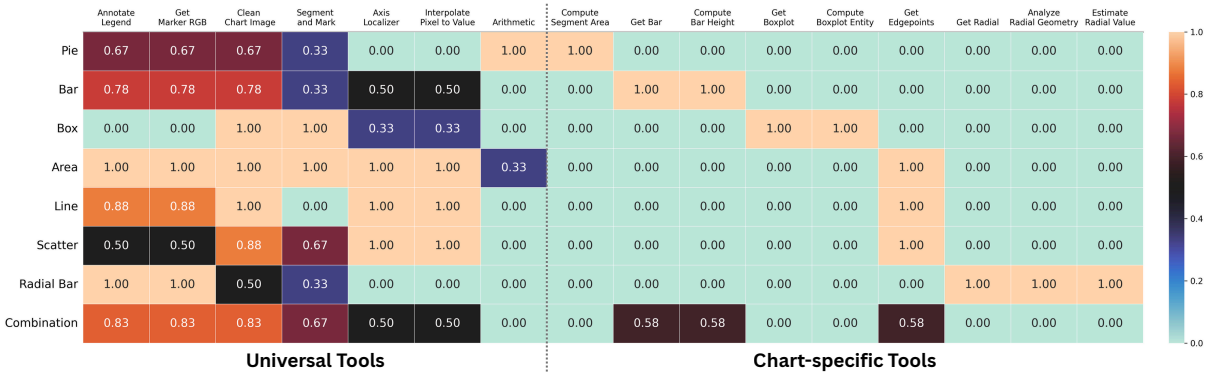


Figure 20: **Tool-use statistics across benchmark datasets.** Percentage of times `ChartAgent` employs a given tool when solving queries for each chart type. As expected, universal tools are used broadly across all chart types, whereas chart-specific tools are invoked selectively depending on the chart type detected by the `ChartAgent` orchestrator.

Table 9: **Ablation study** on the role of tools in chart VQA. Chart-specialized tools enable strong gains, especially for unannotated charts & numeric QA. **Red:** Best.

Method	Tool Type	Overall Acc. (%) ↑	Unannotated & Numeric Acc. (%) ↑
ReAct + No Tools	None	38.84	19.46
ReAct + Natural Image Tools	Generic	41.35	20.50
`ChartAgent` (Ours)	Chart-specialized	**71.39**	**58.29**

This ablation study confirms that generic natural-image tools are fundamentally inadequate for chart reasoning. By equipping the agent with a comprehensive taxonomy of chart-specialized tools, integrated into an iterative ReAct loop with visual self-verification, `ChartAgent` achieves state-of-the-art performance—particularly excelling on unannotated charts and numeric QA where prior methods fail.

K.4 Visual and Reasoning Complexity Analysis

Table 10 presents the accuracy on unannotated charts by visual complexity of the charts and reasoning complexity of the chart–QA pairs.

K.5 Accuracy vs. LLM-as-a-Judge

We found that GPT often *relaxes* the 5% margin condition, leading to inflated performance compared to arithmetic accuracy, which strictly en-

Table 10: **Accuracy by Complexity Levels.** Accuracy (%) on unannotated charts stratified by visual complexity of the charts and reasoning complexity of the chart–QA pairs. **Red:** Best, Blue: Second best.

Model	Visual Complexity			Reasoning Complexity			Overall
	Easy	Medium	Hard	Easy	Medium	Hard	Average ↑
Proprietary Multimodal Large Language Models							
GPT 4o	57.16	28.25	17.59	44.06	20.84	13.72	36.15
GPT 4o-mini	39.93	20.22	9.45	32.06	9.94	9.39	25.19
Claude 3 Haiku	40.53	21.17	10.42	33.17	10.33	9.39	26.04
Gemini 1.5	46.36	20.83	6.19	36.43	9.35	1.08	27.27
Open-weights Multimodal Large Language Models							
BLIP-2	3.16	2.45	4.56	3.06	3.44	1.08	2.92
CogAgent	13.23	11.78	6.51	13.44	8.41	5.78	11.62
CogVLM	15.17	9.94	11.07	12.94	9.18	8.66	11.73
DeepSeek-VL2	43.08	25.39	19.54	37.00	16.63	12.64	30.31
DocOwl1.5-Chat	43.08	15.45	10.10	29.72	10.33	8.66	23.58
InstructBLIP	9.83	4.02	4.56	6.67	3.82	5.05	5.92
InternVL3	49.27	22.67	21.17	37.89	16.83	12.27	30.92
LLama3.2	58.01	28.86	14.33	45.28	14.15	20.58	36.38
Llava1.6	15.66	7.69	5.21	12.78	2.68	5.05	9.92
Llava1.5	8.50	6.19	6.84	7.83	6.31	2.89	7.00
LlaVA-OneVision	11.17	9.39	14.01	11.39	10.71	4.33	10.50
mPLUG-Owl3	18.81	9.67	10.42	14.89	8.99	5.05	12.65
Phi3-vision	55.83	36.08	**22.48**	50.11	19.69	19.49	40.73
Pixtral	45.39	22.94	11.73	35.39	14.53	12.27	28.73
Qwen2VL	66.02	36.69	15.64	54.44	17.40	21.66	43.50
QwenVLChat	8.98	5.65	4.23	7.61	3.25	5.78	6.54
SmolVLM	23.06	10.42	10.75	17.83	9.37	2.17	14.46
SPHINX-V	20.26	8.44	9.44	15.22	7.07	3.24	12.30
VisualGLM	12.74	5.51	4.23	9.56	3.44	3.25	7.65
Chart-related Models							
ChartGemma	39.68	15.66	8.47	28.72	7.46	9.75	22.42
ChartInstruct	38.96	12.05	8.79	25.67	7.27	9.03	20.19
ChartLlama	17.84	9.26	4.56	13.61	5.93	7.58	11.42
ChartVLM	44.90	15.11	9.77	31.56	6.31	7.58	23.92
DePlot	50.36	19.54	9.77	37.78	5.16	9.03	28.15
MatCha	17.48	6.81	2.61	13.22	1.72	1.81	9.69
OneChart	52.21	15.61	5.22	34.17	4.29	2.82	26.81
TinyChart	53.03	24.71	16.94	40.00	16.83	15.88	32.77
UniChart	30.83	10.28	3.26	21.06	3.06	7.22	15.96
Multimodal Agentic Framework							
ChartAgent **(Ours)**	**83.98**	**56.77**	17.92	**71.33**	**41.68**	**28.52**	**60.81**

(a) ChartBench Dataset

Model	Visual Complexity			Reasoning Complexity			Overall
	Easy	Medium	Hard	Easy	Medium	Hard	Average ↑
Proprietary Multimodal Large Language Models							
GPT 4o	42.11	47.77	22.70	49.86	31.54	22.22	39.44
GPT 4o-mini	36.22	39.28	22.09	43.21	24.48	24.07	33.94
Claude 3 Haiku	25.69	32.59	16.56	34.07	17.01	17.59	25.77
Gemini 1.5	36.84	31.25	20.86	44.60	18.26	16.67	31.41
Open-weights Multimodal Large Language Models							
BLIP-2	0.93	1.78	3.07	1.38	2.48	0.93	1.69
CogAgent	26.06	25.89	21.47	30.47	18.67	20.37	24.93
CogVLM	26.62	23.21	20.85	27.98	19.92	21.30	24.23
DeepSeek-VL2	42.41	36.61	20.86	47.37	24.07	22.22	35.63
DocOwl1.5-Chat	28.79	23.66	16.56	32.41	17.43	12.96	24.37
InstructBLIP	8.05	8.04	11.66	8.03	8.71	12.04	8.87
InternVL3	40.25	41.52	22.70	46.26	28.21	23.15	36.62
LLama3.2	49.23	37.95	23.93	49.31	31.95	25.93	39.86
Llava1.6	19.20	18.75	15.33	21.32	13.27	18.52	18.17
Llava1.5	14.55	14.29	14.72	16.34	12.45	12.96	14.51
LlaVA-OneVision	12.69	15.63	9.20	16.89	7.88	10.19	12.82
mPLUG-Owl3	21.67	16.96	13.49	21.33	15.35	14.81	18.31
Phi3-vision	46.74	41.07	32.52	53.74	26.97	34.26	41.69
Pixtral	45.82	39.73	20.86	49.58	27.39	24.07	38.17
Qwen2VL	51.39	40.18	**28.83**	**55.13**	32.37	24.07	42.96
QwenVLChat	19.19	24.10	17.79	23.82	15.76	19.44	20.42
SmolVLM	23.22	25.44	15.33	25.76	18.25	18.52	22.11
SPHINX-V	21.67	20.08	19.63	25.20	16.59	14.81	20.70
VisualGLM	10.52	15.18	15.34	14.68	8.71	17.59	13.10
Chart-related Models							
ChartGemma	31.89	32.58	17.79	37.67	18.25	23.14	28.87
ChartInstruct	22.60	17.85	7.97	24.37	11.61	9.25	17.75
ChartLlama	20.12	22.76	22.69	22.43	19.08	24.07	21.55
ChartVLM	33.13	28.57	21.47	35.45	23.65	19.44	29.01
DePlot	49.22	26.78	15.95	45.70	28.21	11.11	34.51
MatCha	17.95	19.64	11.65	20.77	13.69	12.03	17.04
OneChart	**55.73**	25.35	13.38	45.55	36.77	6.45	37.14
TinyChart	39.93	32.14	22.08	44.04	22.82	21.29	33.38
UniChart	25.69	13.83	12.26	26.31	11.61	10.18	18.87
Multimodal Agentic Framework							
ChartAgent **(Ours)**	50.93	**49.91**	24.54	54.14	**38.17**	**27.78**	**44.16**

(b) ChartX Dataset

forces this threshold. This observation is important to share with the community, as most recent Chart VQA papers (Xu et al., 2023; Xia et al., 2024; Masry et al., 2022) rely directly on GPT-based accuracy for evaluation. Table 11 reports the comparison between our standardized accuracy evaluation and the corresponding LLM-as-a-Judge results on the ChartBench dataset.

Table 11: **Accuracy vs. LLM-as-a-Judge.** Results on the ChartBench dataset. All values represent accuracy in percentage.

Model	Accuracy	LLM-as-a-Judge	Gap (%)
Gemini 2.0 flash	69.90	76.45	-6.55
GPT 4o-mini	42.24	48.47	-6.24
DeepSeek-VL2	49.39	55.16	-5.76
ChartLlama	19.89	24.42	-4.53
ChartInstruct	31.24	35.68	-4.45
GPT 4o	51.47	55.63	-4.16
SPHINX-V	19.76	23.79	-4.03
TinyChart	46.84	50.82	-3.97
CogVLM	28.11	31.68	-3.58
ChartGemma	39.32	42.76	-3.45

K.6 Concurrent Works

The ChartBench dataset was released on December 26, 2023, and ChartX on February 19, 2024. Table 12 shows the split of models with knowledge cutoff dates before versus after each dataset release. Since datasets may have leaked into the training data of models with knowledge cutoff dates after release, we report these concurrent model results separately.

K.6.1 Performance of Concurrent Works on Public Benchmarks

Table 13 presents the accuracy comparison for concurrent works with knowledge cutoff dates after the dataset releases.

We suspect that benchmark data (ChartBench and ChartX, released in December 2023 and February 2024, respectively) may have been included in the training data of GPT-o3 and GPT-o4-mini (knowledge cutoff: May 2024). In several cases, particularly with GPT-o3, we observed that the model produced correct answers despite incorrect

Table 12: **Knowledge Cutoffs and Concurrent Works.** Comparison of model and dataset release dates relative to ChartBench and ChartX, showing whether models were trained before or after these benchmarks.

Model / Dataset	Knowledge Cutoff	Relative to ChartBench / ChartX
Claude 3 Haiku	Aug 1, 2023	Before both
Claude 3 Sonnet	Aug 1, 2023	Before both
GPT-4o	Sept 30, 2023	Before both
GPT-4o-mini	Sept 30, 2023	Before both
GPT-o1	Sept 30, 2023	Before both
ChartBench Dataset	Dec 26, 2023	—
ChartX Dataset	Feb 19, 2024	—
Claude 3.5 Sonnet	Apr 1, 2024	After both
GPT-o3	May 31, 2024	After both
GPT-o4-mini	May 31, 2024	After both
GPT-4.1	May 31, 2024	After both
Claude 3.5 Haiku	Jul 1, 2024	After both
Gemini 2.0	Aug 1, 2024	After both
Claude 3.7 Sonnet	Nov 1, 2024	After both
Mistral-Small	Mar 17, 2025	After both

reasoning steps or tool outputs. For example, even when the agent misidentified key visual elements or generated invalid intermediate outputs, the final answer was still correct. We also noted this behavior in instances where it was humanly very difficult to provide the exact answer, yet GPT-o3 and GPT-o4-mini produced outputs with decimal-level precision. Such patterns suggest possible memorization or exposure to similar instances during training.

While preliminary, these observations provide strong evidence of potential data leakage from public benchmarks into newer models. To strengthen this analysis, we curated a new held-out internal dataset that mirrors the complexity of ChartBench and ChartX, enabling a more rigorous evaluation.

K.6.2 Performance of Concurrent Works on the Internal Dataset

We created a new dataset with 125 chart–QA pairs that we are confident were not included in the training data of newer models, and conducted evaluations for a fairer comparison of these models against `ChartAgent`. Specifically, we collected unannotated charts such as bar, line, pie, and bar–line combinations requiring numeric QA from the open web, selecting only those whose ground-truth answers are unavailable online, thereby increasing confidence that they were not included in the training data of newer models.

Table 14 reports the overall accuracy (within a 5% margin) and average numeric error on this curated dataset. Clearly, `ChartAgent` outperforms all newer models by a significant margin in both accuracy and average error, achieving a +10.48% absolute accuracy gain over the second-best model (GPT-5) and a 5.72-point reduction in average absolute error relative to GPT-o3. Notably, the baselines include both recent closed-source models (e.g., GPT-5) and agentic variants (e.g., o3 and o4-mini). These results further reinforce `ChartAgent`'s effectiveness as a chart-focused visually-grounded reasoning framework.

K.7 Visual Self-Verification and Recovery Behavior

In addition to analyzing difficulty-based trends, we studied whether `ChartAgent` could detect unsatisfactory tool outputs and recover using its visual self-verification mechanism. We manually evaluated 30 randomly selected agent trajectories from the ChartBench dataset to assess this behavior. The results are summarized in Table 15. In 50% of the sampled cases, the tool outputs were correct, and no recovery was needed. In the remaining 50%, the agent correctly identified the tool outputs as unsatisfactory and triggered its self-verification mechanism. Among these, 70% resulted in successful recovery, leading to correct final answers. The remaining 30% failed to recover, contributing to a 15% overall error rate attributable to unresolved tool-level failures. These findings demonstrate that `ChartAgent`'s visual self-verification mechanism is both frequently invoked and often effective, enhancing robustness in the presence of imperfect tool outputs—especially critical for unannotated chart understanding.

K.8 Fallback Analysis: When `ChartAgent` Reverts to the Base Model and Common Trigger Conditions

We conducted a manual analysis of 30 randomly selected agent trajectories from ChartBench, focusing on unannotated charts and numeric QA, to better understand when and why the agent reverts to the base model (GPT-4o). We found that the fallback rate was relatively low—less than 10% across the sample. The most common reasons for fallback included the following:

- Bar charts: When the computed bar height was negative or highly inconsistent with the axis values, indicating a failure in visual estimation, the agent abandoned tool-based reasoning and allowed GPT-4o to attempt a direct response.
- OCR-based tools returning `None`: For example,

Table 13: **Accuracy on Concurrent Works (Public Benchmarks).** Comparison of accuracy (%) on concurrent works with knowledge cut-off dates after the release of the datasets. All values correspond to the highest performance achieved across zero-shot and CoT prompting styles for each MLLM. Ann./Unann. denote Annotated and Unannotated charts. RL QA: Relationship QA; VC/GC QA: Value Comparison & Global Conception QA.

Model	Chart Types		Question Types		Overall
	Ann.	Unann.	Numeric QA	RL QA	Avg. ↑
Proprietary Multimodal Large Language Models					
GPT o3	98.18	76.56	82.55	98.44	83.39
GPT o4-mini	98.50	71.73	79.14	99.00	80.18
GPT 4.1	97.33	67.00	75.61	94.00	76.58
Gemini 2.0 flash	97.79	58.31	71.81	41.00	69.90
Claude 3.7 Sonnet	97.75	60.38	71.64	82.00	72.18
Claude 3.5 Sonnet	96.50	56.23	68.14	83.50	68.95
Claude 3.5 Haiku	90.67	38.58	53.89	75.50	55.03
Open-weights Multimodal Large Language Models					
Mistral	91.75	43.23	57.08	90.00	58.55
Multimodal Agentic Framework					
ChartAgent (**Ours**)	**94.33**	**60.81**	**70.91**	**91.00**	**71.39**

(a) ChartBench Dataset

Model	Chart Types		Question Types		Overall
	Ann.	Unann.	Numeric QA	VC/GC QA	Avg. ↑
Proprietary Multimodal Large Language Models					
GPT o3	91.18	71.13	79.59	76.85	78.82
GPT o4-mini	91.18	72.68	80.92	76.85	79.77
GPT 4.1	92.99	69.58	77.90	80.25	78.56
Gemini 2.0 flash	89.37	58.31	68.72	74.07	70.23
Claude 3.7 Sonnet	89.37	60.28	69.81	75.62	71.44
Claude 3.5 Sonnet	87.78	57.32	67.39	73.15	69.01
Claude 3.5 Haiku	80.32	40.70	50.97	68.52	55.90
Open-weights Multimodal Large Language Models					
Mistral	84.84	48.59	59.06	71.30	62.50
Multimodal Agentic Framework					
ChartAgent (**Ours**)	**84.84**	**44.16**	**55.93**	**69.14**	**59.69**

(b) ChartX Dataset

Table 14: **Accuracy on Concurrent Works (Internal Benchmarks).** Overall average accuracy (within 5% margin) and average error across models on the curated internal dataset. **Red:** Best, Blue: Second best.

Model	Accuracy (%) ↑	Avg. Error (%) ↓
ChartAgent (ours)	**85.19**	**3.42**
GPT 5	74.71	24.09
GPT 5-mini	73.18	11.24
Claude 3.7 Sonnet	69.71	15.52
GPT o4-mini	69.68	21.88
Gemini 2.0	67.24	21.07
GPT-4.1	66.61	24.32
GPT-o3	62.93	9.14
Claude 3.5 Haiku	42.11	37.31
Mistral	38.54	38.74
o1	33.07	44.31
GPT-4o	22.02	64.34

Table 15: Visual self-verification and recovery outcomes in `ChartAgent` trajectories.

Metric	Value
Cases where recovery was needed (i.e., tool output deemed unsatisfactory)	50%
Successful recoveries among needed cases	**70%**
Correct final answers following recovery	**70%**
Cases where tool error propagated to final answer (i.e., remained incorrect)	15%

if legend or axis label detection failed to locate any relevant entities, the agent deemed the output unsatisfactory and reverted to GPT-4o.

• Line charts: When edge-point detection or interpolation tools produced empty outputs or values that were highly inconsistent with the axis, the agent once again defaulted to GPT-4o.

In all such cases, the agent judged tool-based reasoning to be unreliable and defaulted to the base model. While rare, this fallback mechanism serves as a valuable fail-safe.

K.9 Inference Time Comparison: `ChartAgent` vs. GPT-4o

We conducted a preliminary timing analysis on a representative subset of samples across various chart types to compare the inference time between `ChartAgent` and GPT-4o. On average:

- A single GPT-4o call with chain-of-thought reasoning took approximately 6-10 seconds per query.
- In contrast, a full `ChartAgent` trajectory, including multi-step tool invocations and self-verification, took approximately 90 seconds per query in the non-parallelized setting, and about 30 s when parallelizable steps were executed concurrently. For comparison, OpenAI's agentic model o3 took 25–40 s on the same task, even when answers were inaccurate.

This increase in inference time is expected due to the agentic design, which involves iterative reasoning, multiple visual perception tool executions, and verification steps. We note that the runtime can be significantly reduced in practice by optimizing tool

efficiency—many intermediate outputs currently used for visualization and debugging can be streamlined or skipped entirely in deployment settings. Despite the additional overhead, we believe the substantial accuracy improvements, particularly on unannotated and numerically dense charts, justify the increased computational cost in settings where precision is critical.

K.10 Monetary Cost Analysis

Our approach incurs monetary costs due to the use of OpenAI's GPT-4o (Hurst et al., 2024) as the base reasoning model. We spent approximately $2000 to run `ChartAgent` on both datasets, covering 4952 chart image and QA pairs across diverse chart types—resulting in an average cost of approximately $0.40 per sample. This cost can be substantially reduced by using smaller models such as GPT-4o-mini, or eliminated entirely with open-source models like Pixtral, Llama, or Qwen, since our framework is designed to be plug-and-play. For example, switching from GPT-4o to GPT-4o-mini would reduce the average cost per sample by more than $15\times$ (to roughly $0.025), making large-scale evaluation far more economical. Thus, monetary cost should not be considered a serious limitation, as our approach can seamlessly adapt to free or low-cost models as well.

L Details on Failure Mode Analysis

`ChartAgent` encounters two main categories of failure: visual perception challenges and reasoning ambiguities.

1) **Perception-based failures.**

(1.1) *OCR obstruction by visual overlays:* Black overlays or dense chart elements often cover axis or legend text, preventing accurate OCR extraction.

(1.2) *Poor color contrast:* Labels in white placed over fluorescent yellow or similarly bright backgrounds are difficult for vision tools to detect.

(1.3) *Legend occlusion:* In some charts, the legend overlaps with key visual elements—such as bars of interest—hindering accurate region detection.

(1.4) *Chart element invisibility:* Median lines in box plots that share the same color as the box become indistinguishable, making it hard to extract correct values.

(1.5) *Segmentation failure due to axis overlap:* Axis lines overlapping with chart elements confuse the segmentation tool and result in incorrect extraction.

(1.6) *Overlap-induced indistinguishability:* When multiple data series substantially overlap in charts (e.g., radar plots, line charts, scatterplots with dense clusters, or filled regions), subtle differences between categories become imperceptible. This occurs due to coincident paths, stacked fills, or saturation effects, preventing reliable detection of fine-grained deviations.

(1.7) *Axis interpretation failures:* When unusual or complex axes (e.g., 3D distorted axes, multiple Y-axes with different scales) make it visually hard to map chart elements to the correct reference values.

2) **Reasoning-based failures.**

(2.1) *Unit mismatches:* The agent sometimes multiplies values based on axis labels (e.g., reading 160 as 160,000 due to "in thousands"), which may not match the ground truth.

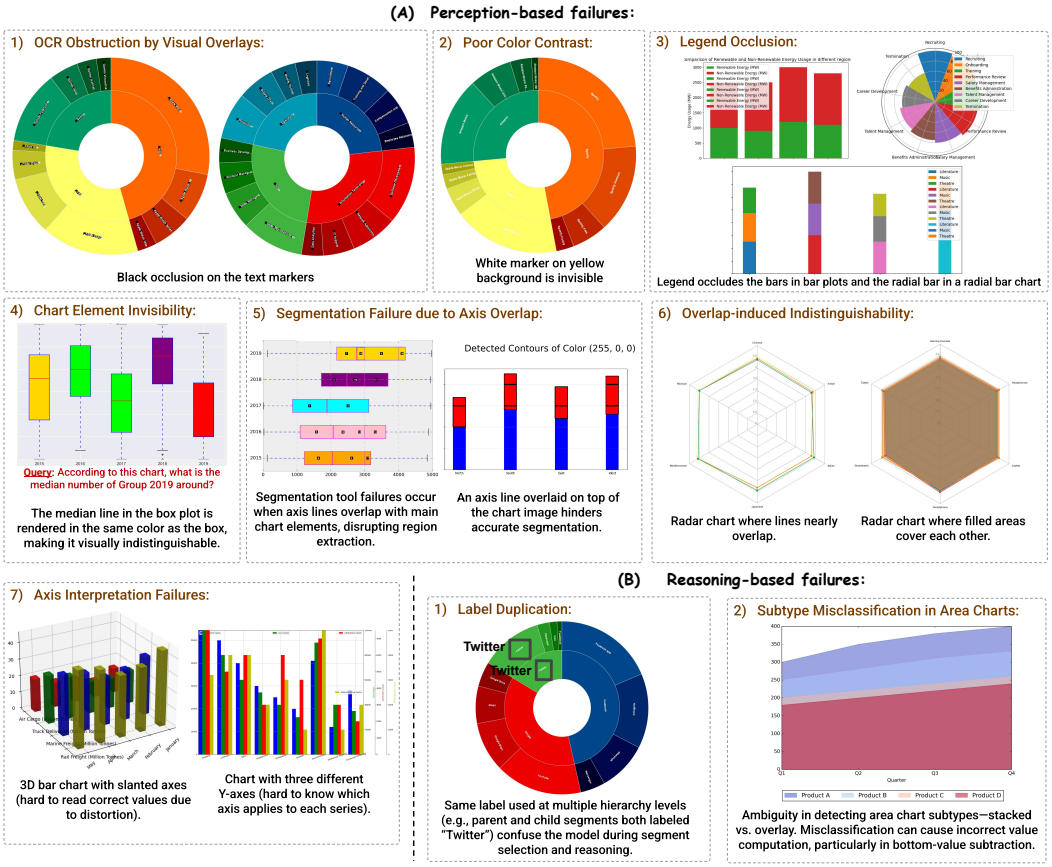
(2.2) *Incorrect tool selection:* Occasionally, the agent chooses the wrong measurement tool—for instance, computing area instead of height—leading to incorrect results despite correct region localization.

(2.3) *Question ambiguity:* Some questions, such as those from multi-ring pie charts in ChartBench, lack clear context (e.g., undefined denominators), resulting in ambiguous interpretation. We plan to address such cases in future work by enabling the agent to detect ambiguity and proactively request user clarification when necessary.

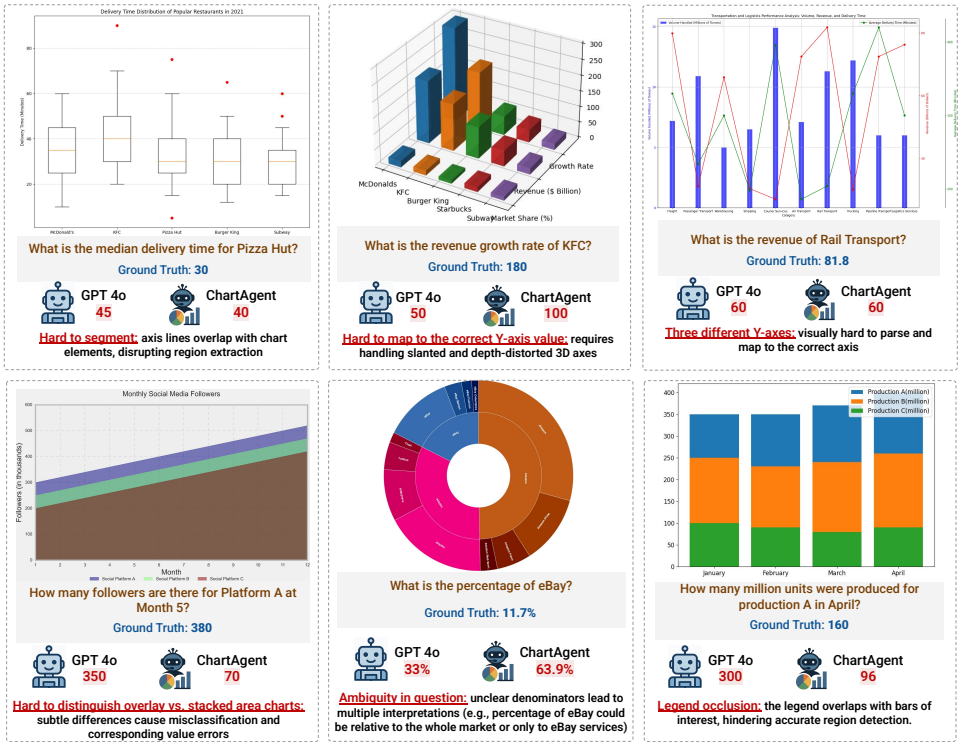
(2.4) *Label duplication:* Charts with the same label used at multiple hierarchy levels (e.g., parent and child segments both labeled "Netflix") confuse the model during segment selection and reasoning. See Appendix L for examples.

(2.5) *Subtype misclassification in area charts:* Overlay and stacked area charts can appear visually similar, and misclassifying them leads to incorrect answer logic (e.g., value subtraction errors), even if all other steps are executed correctly

See Figure 21 for illustrations of common failure modes (21a) and qualitative failure cases where `ChartAgent` produces incorrect responses (21b). Overall, most failures are perception-driven, originating from chart tool errors rather than complex reasoning or planning.



(a) **Illustrations of common failure modes in `ChartAgent`.**



(b) **Qualitative failure cases where `ChartAgent` produces incorrect responses.**

Figure 21: **Failure Mode Analysis.** Examples where `ChartAgent` fails to produce the correct response due to visual perception challenges or reasoning ambiguities. *(A) Perception-based failures* include OCR obstruction by overlays, poor color contrast, key chart element occlusions (e.g., legends blocking bars), chart element invisibility, difficult segmentation (e.g., overlapping axes or cluttered regions), overlap confusion, 3D depth distortion, and multiple Y-axis mapping errors. *(B) Reasoning-based failures* include label duplication, ambiguous questions (e.g., undefined denominators) and misclassification of visually similar chart subtypes (e.g., stacked vs. overlay area).

M Prompts

We present the prompts used for `ChartAgent` M.1, baselines M.2, evaluation M.3, and complexity analysis M.4. Note that some low-level prompt details are omitted below for space constraints. The complete prompts are available in our repository at: `https://anonymous.4open.science/r/ChartAgent`.

M.1 ChartAgent Prompts

`ChartAgent` comprises a structured set of prompts that specify reasoning, tool usage, metadata extraction, and in-context learning (ICL). For clarity, we first present the overall concatenated prompt, followed by its individual components: the System Prompt (M.1.1), Chart Tool Definitions (M.1.2), Chart Metadata Extraction Prompt (M.1.3), and ICL Examples (M.1.4).

ChartAgent Prompt

SYSTEM PROMPT [M.1.1]

Instruction:
To support your analysis, several Python-based tools are available in `tools.py` and will be pre-imported for you.

- Bounding boxes follow the format $[x, y, w, h]$, where x and y denote the horizontal and vertical coordinates of the upper-left corner, and w and h represent the width and height of the box.
- Use the provided tools for precise numeric analysis by extracting properties such as area, height, and other quantitative attributes of chart components.
- Execute one tool at a time, and wait for its output before proceeding. If the output seems uncertain, you may re-run the tool with adjusted parameters or switch to a different tool.

Below are the tools defined in `tools.py`:

TOOL DEFINITIONS PROMPT [M.1.2]

—

Below are examples demonstrating how to use the tools to address user requests. You may refer to them for guidance.

IN CONTEXT LEARNING EXAMPLES PROMPT [M.1.4]
Note for readers: Only ICL examples corresponding to the chart type detected in the chart metadata extraction stage are retrieved and used. For example, if the chart is detected as a pie chart, only pie chart ICL examples are included.

—

GOAL:
Using the tools above, reason about how to solve `# USER REQUEST #` and generate step-by-step actions (each action is a Python Jupyter notebook code block) to solve the request.

- You may need to use the tools above to process chart images and numerical values, and to make decisions based on the visual and numerical outputs of previous code blocks.
- The Jupyter notebook has already executed the following code to import the necessary packages:

  ```python
  from PIL import Image
  from IPython.display import display
  from tools import *
  ```

- The generated actions should fully resolve the user request `# USER REQUEST #`. Assume the request is reasonable and solvable; do your best to solve it.
- If you believe you have the answer, output `ANSWER: <your answer>` and end with `TERMINATE`.

—

Here is the chart metadata:

CHART METADATA [M.1.3]
Note for readers: Extracted during the chart metadata extraction stage using the Chart Metadata Extraction Prompt (M.1.3).

Input:

<chart image> # USER IMAGE #: {entry["image"]}
<question> # USER REQUEST #: {entry["query"]}

Instruction (continued):

Now please generate only THOUGHT 0, and ACTION 0 in RESULT. If no action needed, also reply with ANSWER: <your answer> and ends with TERMINATE in the RESULT.

RESULT #:

Agent:

THOUGHT 0: I should annotate the legend in the chart image to correctly identify the marker of interest.
ACTION 0:

```python
legend_image, labeled_legend, bbox_map = annotate_legend(chart_image, legend_texts)
display(labeled_legend)
print (bbox_map)
```

Instruction (continued):

OBSERVATION: Execution success. The output is as follows: <output>

Please generate the next THOUGHT and ACTION. If you can get the answer, please also reply with ANSWER: <your answer> and end with TERMINATE.

Trajectory continues as the agent generates THOUGHT 1, ACTION 1, ... *until termination.*

For visualization purposes, prompts containing code are formatted differently here; in practice, however, all prompts are provided as plain text inputs to the MLLM. The same prompt template is used across all chart VQA samples and datasets in this work.

M.1.1 System Prompt

The system prompt establishes the agent's role and high-level objectives. It instructs the model to follow structured reasoning, invoke tools where appropriate, and return answers in a well-defined format.

System Prompt

You are a helpful multimodal AI assistant for answering questions about chart images, including numeric QA, free-form QA, and multiple-choice QA.

You operate in a Python Jupyter notebook environment and can:

- Suggest Python code (in executable code blocks) to process images, text, or data.
- Use variables and states from previous cells.
- Provide complete code, not partial snippets.
- The notebook imports Image from the PIL package and display from the IPython.display package. Display all image outputs using display().

[MORE INSTRUCTIONS ...] (The actual system prompt contains additional detailed guidelines.)

Execution Instructions:

- Execute one tool at a time and wait for results.
- If an error occurs, fix it and re-run.
- If unsure about output, try different parameters or tools.

For each turn, you should first do a **"THOUGHT"**, based on the chart images and textual question you see. If you think you get the answer to the initial user request, you can reply with: **ANSWER: <your answer>** and end with **TERMINATE**.

M.1.2 Chart Tool Definitions

The following are the Python-based tools available to `ChartAgent`, along with their inputs, outputs, and expected behaviors. An abridged parameter set is shown for some tools to save space and aid readability.

Tool Definitions Prompt

```python
```

Universal Tools

def annotate_legend (image: PIL.Image.Image, legend: dict[str]) → tuple[PIL.Image.Image, PIL.Image.Image, dict[int, tuple[str, tuple[int,int,int,int]]]]:

"""" Detects legend coordinates, crops the legend, and annotates it with numeric labels.

Args:
`image` (`PIL.Image.Image`): Input chart image
`legend` (`dict[str]`): Legend strings

Returns:
`legend_image` (`PIL.Image.Image`): Cropped legend image
`labeled_legend` (`PIL.Image.Image`): Annotated legend image with numeric label mappings
`bbox_mapping` (`dict[int, (str, (int, int, int, int))]`): Maps numeric labels to (text, bounding box coordinates [x_min,y_min,x_max,y_max]).

Example:

```
image = PIL.Image.open("chart_image.png")
legend_image, labeled_legend, bbox_mapping = annotate_legend(image,
legend={"Legend1", "Legend2"})
display(labeled_legend)
```

"""

def get_marker_rgb (image: PIL.Image.Image, bbox_mapping: dict[int, tuple[str, tuple[int, int, int, int]]], text_of_interest: str, label_of_interest: int, distance_between_text_and_marker: int) → tuple[int, int, int]:

"""" Retrieves the dominant RGB color of a legend marker, either by label (from an annotated legend image) or by associated text.

Args:
`image` (`PIL.Image.Image`): Input legend image
`bbox_mapping` (`dict`): Mapping of label numbers to (text, bbox) tuples. The bounding box is (x_min, y_min, x_max, y_max).
`text_of_interest` (`str`, optional): The legend text whose marker color should be retrieved. If provided, fuzzy matching is applied.
`label_of_interest` (`int`, optional): The label number in `bbox_mapping` whose marker color should be retrieved.
`distance_between_text_and_marker` (`int`): Approximate distance in pixels between the legend text and its marker (default: 5).

Returns:
`tuple[int, int, int]`: The (R, G, B) color of the detected marker.

Examples:

```
# Example 1: Using text label
image = Image.open("chart_image.png")
legend_image, labeled_legend, bbox_mapping = annotate_legend(image)
rgb_color = get_marker_rgb(legend_image, bbox_mapping, text_of_interest="Rock")
print("Detected RGB color:", rgb_color) # Output: (0, 0, 255)

# Example 2: Using label number
rgb_color = get_marker_rgb(legend_image, bbox_mapping, label_of_interest=5)
print("Detected RGB color:", rgb_color) # Output: (255, 0, 0)
```

"""

def clean_chart_image (image: PIL.Image.Image, title: str, legend: dict[str]) → PIL.Image.Image:

"""" Cleans a chart image by removing title and legend if provided.

Args:

image (PIL.Image.Image): Input PIL image of the chart
title (str): Title to remove (None to skip title removal)
legend (dict[str]): Legend strings to remove (None to skip legend removal)
Thresholds and expand values control removal bounding boxes.

Returns:
cleaned_image (PIL.Image.Image): Cleaned chart image

Example:

```
image = PIL.Image.open("chart_image.png")
cleaned_image = clean_chart_image(image, title="Title", legend={"Legend1", "Legend2"})
display(cleaned_image)
"""
```

def segment_and_mark (image: PIL.Image.Image, segmentation_model: str, min_area: int, iou_thresh_unique: float, iou_thresh_composite: float, white_ratio_thresh: float, remove_background_color: bool) → tuple[PIL.Image.Image, list[dict]]:

""" Segments an input image using the specified model and applies post-processing to clean the masks through a multi-step filtering pipeline that removes small, duplicate, composite, and background-dominated masks. Returns a labeled image with drawn contours and optional numbered labels, along with a cleaned list of segmentation masks.

Args:
image (PIL.Image.Image): The input chart image to be segmented
segmentation_model (str): Segmentation model (Segment Anything ("SAM") by default)
min_area (int): Minimum pixel area to keep a mask (5000 default)
iou_thresh_unique (float): IoU threshold for duplicate removal (0.9 default)
iou_thresh_composite (float): IoU threshold for composite mask removal (0.98 default)
white_ratio_thresh (float): White pixel ratio to discard mask (0.95 default)
remove_background_color (bool): If True, remove background color pixels

Returns:
labeled_image (PIL.Image.Image): Segmented and labeled image with drawn contours and numeric labels
cleaned_masks (list[dict]): Cleaned list of segmentation masks

Example:

```
image = Image.open("chart_image.png")
labeled_image, cleaned_masks = segment_and_mark(image)
display(labeled_image)
print(f"Total masks: {len(cleaned_masks)}")
"""
```

def axis_localizer (image: PIL.Image.Image, axis: str, axis_threshold: float, axis_tickers: list) → tuple[list[float], list[int]]:

""" Localizes the specified axis (x-axis, left y-axis, or right y-axis) by detecting its numeric tick values and mapping them to corresponding pixel positions in the chart image. Uses Tesseract OCR and EasyOCR.

Args:
pil_image (PIL.Image.Image): Input chart image
axis (str): Axis to localize; 'x' (x-axis), 'y' (left y-axis), or 'right_y' (right y-axis)
axis_threshold (float): Fraction of the image to scan for tick labels along the axis direction (0.2 default)
axis_tickers (list or None): Optional pre-supplied axis tick strings to improve matching

Returns:
axis_values (list[float]): Detected numeric tick values (e.g., [0, 200, 400, 600])
axis_pixel_positions (list[int]): Corresponding pixel positions (e.g., [950, 850, 750, 650])

Example:

```
axis_values, axis_pixel_positions = axis_localizer(image, axis='y', axis_threshold=0.2,
    axis_tickers=["200", "400", "600", "800", "1000", "1200", "1400"])
print(axis_values, axis_pixel_positions)
"""
```

def interpolate_pixel_to_value (pixel: float, axis_values: list[float], axis_pixel_positions: list[int]) → float:

""" Maps a pixel coordinate to its corresponding axis value using linear interpolation between known axis ticks and

their pixel positions.

Args:
`pixel` (`float` or `int`): Pixel coordinate to map
`axis_values` (`list[float]`): Numeric axis values (e.g., `[0, 200, 400, 600]`)
`axis_pixel_positions` (`list[int]`): Pixel positions corresponding to `axis_values` (e.g., `[950, 850, 750, 650]`)

Returns:
`float`: Interpolated axis value corresponding to the given pixel

Example:

```
axis_values = [0, 200, 400, 600]
axis_pixel_positions = [950, 850, 750, 650]
val = interpolate_pixel_to_value(800, axis_values, axis_pixel_positions)
print(val) # Expected interpolation between 200 and 400
```

"""

def arithmetic (a: float, b: float, operation: str) → float:

""" Performs a specified arithmetic operation between two numeric inputs. Supports operations such as addition, subtraction, multiplication, division, percentage, and ratio.

Args:
`a` (`float`): First operand
`b` (`float`): Second operand
`operation` (`str`): Arithmetic operation to perform. Supported: `"add"`, `"subtract"`, `"multiply"`, `"divide"`, `"percentage"`, `"ratio"` (`"percentage"` by default)

Returns / Raises:
`float`: Result of the arithmetic operation
`ValueError`: If an unsupported operation is provided or division by zero occurs

Example:

```
total = 1200
part = 300
result = arithmetic(part, total, operation="percentage")
print("Percentage:", result) # Output: 25.0
```

"""

Chart-specific Tools

Pie Chart | Treemap

def compute_segment_area (image: PIL.Image.Image, filter_rgb: tuple[int,int,int], measure: str, masks: list, filter_segment: list) → tuple[PIL.Image.Image, int]:

""" Computes the area of a chart segment by: (1) counting discrete visual elements of a specified color, (2) counting pixels of a specified color, or (3) counting pixels within a segment identified by a specific label ID. Commonly used for pie charts and tree maps.

Args:
`image` (`PIL.Image.Image`): Input chart image (cleaned if necessary)
`filter_rgb` (`tuple[int,int,int]`, optional): RGB values to filter by; if `None`, uses full chart
`measure` (`str`): Method to measure area — `"pixels"` or `"discrete-dots"`
`masks` (`list`, optional): Segmentation masks (SAM-style)
`filter_segment` (`list`, optional): Segment label numbers to include in pixel counting

Returns / Raises:
`visualization` (`PIL.Image.Image`): Image with detected/filtered areas highlighted
`int`: Computed area (discrete-dots or pixels)
`ValueError`: If `measure` is unsupported

Examples:

```
    # Example 1: Full pie chart area (discrete-dots)
    image = Image.open("pie_chart.png")
    vis, area = compute_segment_area(image, measure="discrete-dots")
    print(area) # e.g., 500

    # Example 2: Area of RGB-colored section (pixels)
    rgb_interest = (255, 0, 0) # red
    vis, area = compute_segment_area(image, filter_rgb=rgb_interest, measure="pixels")
    print(area) # e.g., 5000

    # Example 3: Area of specific segments via masks
    labeled_img, masks = segment_and_mark(image)
    vis, area = compute_segment_area(image, measure="pixels", masks=masks, filter_segment
        =[3,5,7])
    print(area) # e.g., 8453
  """
```

Bar Chart

def get_bar (image: PIL.Image.Image, rgb_of_interest: tuple[int,int,int], ticker_label: str, segmentation_model: str, bar_orientation: str) → tuple[int, int, int, int]:

""" Detects and returns the bounding box of a bar in a chart image that matches a specified color and/or axis label. It segments bar regions using a model, filters by color if provided, locates the target axis label using OCR if specified, and selects the closest matching bar accordingly. Commonly used for bar charts.

Args:
- `image` (`PIL.Image.Image`): Input chart image
- `rgb_of_interest` (`tuple[int,int,int]`, optional): RGB color of target bar
- `ticker_label` (`str`, optional): Axis label text of interest
- `segmentation_model` (`str`): Segmentation model for detection (`"SAM"` default)
- `bar_orientation` (`str`): `"vertical"`, `"horizontal"`, or `"vertical-right"` (`"vertical"` default)

Returns:
- `tuple[int,int,int,int]`: Bounding box `(x, y, w, h)` if bar is found, else `None`

Examples:

```
    # Example 1: Vertical bar plot
    image = Image.open("bar_chart.png")
    bbox = get_bar(image, rgb_of_interest=(100,128,45), ticker_label="2016")
    print(bbox) # e.g., (50, 100, 30, 200)

    # Example 2: Combination bar-line plot
    bbox = get_bar(image, rgb_of_interest=(100,128,45), ticker_label="2016", bar_orientation
        = "vertical-right")
    print(bbox)
  """
```

def compute_bar_height (image: PIL.Image.Image, bar_of_interest: tuple[int,int,int,int], bar_orientation: str, axis_threshold: float, x_axis_tickers: list, y_axis_tickers: list, x_axis_title: str, y_axis_title: str) → float:

""" Computes a bar's value (height or length) by mapping its pixel bounding box to axis values using OCR-based axis localization. Supports left/right y-axes for vertical bars and the x-axis for horizontal bars. Commonly used for bar charts.

Args:
- `image` (`PIL.Image.Image`): Input chart image
- `bar_of_interest` (`tuple[int,int,int,int]`): Bounding box `(x, y, w, h)` of the bar
- `bar_orientation` (`str`): `"vertical"`, `"vertical-right"`, or `"horizontal"` (`"vertical"` default)
- `axis_threshold` (`float`): Fraction of the image scanned for tick labels during axis localization (`0.15` default)
- `x_axis_tickers` (`list` or `None`): Optional pre-read x-axis tick labels
- `y_axis_tickers` (`list` or `None`): Optional pre-read y-axis tick labels
- `x_axis_title` (`str` or `None`): X-axis title, if available
- `y_axis_title` (`str` or `None`): Y-axis title, if available

Returns:
- `float`: Estimated bar value (height for vertical; length for horizontal)

Examples:

```
# Example 1: Vertical bar on left y-axis
image = Image.open("bar_chart.png")
bar = (120, 210, 35, 180) # (x, y, w, h) from get_bar()
bar_height = compute_bar_height(image, bar, bar_orientation="vertical")
print(bar_height)

# Example 2: Horizontal bar (value from x-axis)
bar = (100, 70, 150, 25)
bar_length = compute_bar_height(image, bar, bar_orientation="horizontal", x_axis_tickers
    =["200", "400", "600", "800", "1000", "1200", "1400"])
print(bar_length)
```

"""

Box Plot

def get_boxplot (image: PIL.Image.Image, masks: list, rgb_of_interest: tuple[int,int,int], ticker_label: str, box_labels_of_interest: list, boxplot_orientation: str, axis_threshold: float) → list:

""" Detects and returns boxplot segments filtered by color, axis label, or segmentation indices. Handles both horizontal and vertical boxplot orientations and supports fuzzy matching for axis-aligned labels and approximate color filtering. Commonly used for box plots.

Args:
`image` (`PIL.Image.Image`): Input chart image
`masks` (`list`): List of segmentation masks
`rgb_of_interest` (`tuple[int,int,int]` or `None`): RGB color to filter segments
`ticker_label` (`str` or `None`): Axis label (e.g., "Tuesday") to filter segments
`box_labels_of_interest` (`list` or `None`): Segmentation mask indices to select
`boxplot_orientation` (`str`): `"vertical"` or `"horizontal"` (`"vertical"` default)
`axis_threshold` (`float`): Fraction of image scanned for axis values (`0.15` default)

Returns:
`list[tuple[int,int,int,int]]`: Final filtered `(x, y, w, h)` segments

Examples:

```
# Example 1: Filter by RGB color (vertical boxplot)
image = Image.open("box_plot.png")
boxplot_of_interest = get_boxplot(image, masks=masks, rgb_of_interest=(106, 184, 209))
print(boxplot_of_interest)

# Example 2: Filter by ticker label (vertical boxplot)
boxplot_of_interest = get_boxplot(image, masks=masks, ticker_label="Tuesday")
print(boxplot_of_interest)

# Example 3: Filter by segmentation indices (horizontal boxplot)
boxplot_of_interest = get_boxplot(image, masks=masks, box_labels_of_interest=[3, 7],
    boxplot_orientation="horizontal")
print(boxplot_of_interest)
```

"""

def compute_boxplot_entity (image: PIL.Image.Image, boxplot_of_interest: list[tuple[int,int,int,int]], boxplot_orientation: str, entity_of_interest: str, axis_threshold: float, x_axis_tickers: list, y_axis_tickers: list) → float:

""" Computes a statistical entity (e.g., max, min, median, Q1, Q3, range, or interquartile range) of a boxplot by mapping its pixel coordinates to value space using axis localization. Commonly used for box plots.

Args:
`image` (`PIL.Image.Image`): Input chart image
`boxplot_of_interest` (`list[tuple[int,int,int,int]]`): Bounding boxes of the boxplot segments
`boxplot_orientation` (`str`): `"vertical"` or `"horizontal"` (`"vertical"` default)
`entity_of_interest` (`str`): One of `"median"`, `"max"`, `"min"`, `"range"`, `"iqr"`, `"q1"`, `"q3"`, `"q2"` (`"median"` default)
`axis_threshold` (`float`): Fraction of image scanned for tick labels during axis localization (`0.15` default)
`x_axis_tickers` (`list` or `None`): Optional pre-read x-axis tick labels
`y_axis_tickers` (`list` or `None`): Optional pre-read y-axis tick labels

Returns:
`float`: Computed value of the requested boxplot entity

Examples:

```
# Example 1: Median value of vertical boxplot
image = Image.open("box_plot.png")
boxplot_segments = [(120, 150, 40, 80), (120, 250, 40, 70)] # from get_boxplot()
median_val = compute_boxplot_entity(image, boxplot_segments, entity_of_interest="median")
print(median_val)

# Example 2: Maximum value (Q1) of horizontal boxplot
boxplot_segments = [(100, 70, 120, 30), (250, 70, 140, 30)]
max_val = compute_boxplot_entity(image, boxplot_segments, "horizontal", "max")
print(max_val)

# Example 3: Interquartile range (IQR) of vertical boxplot
boxplot_segments = [(130, 160, 30, 70), (130, 260, 30, 90)]
iqr_val = compute_boxplot_entity(image, boxplot_segments, entity_of_interest="iqr")
print(iqr_val)
"""
```

Line | Area | Scatter Plots

def get_edgepoints (image: PIL.Image.Image, masks:list, rgb_of_interest: tuple[int,int,int], ticker_label: str, mask_labels_of_interest: list, chart_orientation: str, lineplot_get_dot: bool, axis_threshold: float) → list[tuple[int,int]]:

""" Computes edge points of a chart segment filtered by color, axis label, or segmentation indices. The edge is determined by scanning perpendicular to the center of the matched label. Supports both vertical and horizontal chart orientations and handles lineplot dots. Useful for identifying segment bounds for downstream value extraction. Commonly used for line, area, and scatter plots.

Args:
`image` (`PIL.Image.Image`): Input chart image
`masks` (`list` or `None`): SAM masks with `bbox` and `segmentation` fields
`rgb_of_interest` (`tuple[int,int,int]` or `None`): Target RGB color for filtering
`ticker_label` (`str` or `None`): Axis label (e.g., "Q3") for filtering
`mask_labels_of_interest` (`list` or `None`): SAM mask indices to select
`chart_orientation` (`str`): `"vertical"` or `"horizontal"` (`"vertical"` default)
`lineplot_get_dot` (`bool`): Whether to get edge points for lineplot dots (`True`) or area chart segments (`False`) (`False` default)
`axis_threshold` (`float`): Portion of the image scanned for axis localization (`0.15` default)

Returns:
`list[tuple[int,int]]`: Edge points perpendicular to the label center:
- Vertical: `[(x, top_y), (x, bottom_y)]`
- Horizontal: `[(left_x, y), (right_x, y)]`

Examples:

```
# Example 1: RGB color + ticker label (vertical area chart)
image = Image.open("area_chart.png")
edge_points = get_edgepoints(image, rgb_of_interest=(106, 184, 209), ticker_label="Q2")
print(edge_points)

# Example 2: SAM mask index + ticker label (vertical area chart)
edge_points = get_edgepoints(image, masks, ticker_label="A", mask_labels_of_interest=[3])
print(edge_points)

# Example 3: RGB color + ticker label (line plot)
image = Image.open("line_plot.png")
line_dots = get_edgepoints(image, rgb_of_interest=(237, 0, 209), ticker_label="Q1",
    lineplot_get_dot = True)
print(line_dots)
"""
```

Radial Bar Plot

def get_radial (image: PIL.Image.Image, rgb_of_interest: tuple[int,int,int], ticker_label: str, segmentation_model: str) → tuple[int,int,int,int]:

""" Computes the coordinates for the radial bar segment of interest using either color-based filtering or segmentation mask labels. Commonly used for radial bar plots.

Args:
`image` (`PIL.Image.Image`): Input chart image
`rgb_of_interest` (`tuple[int,int,int]` or `None`): Target RGB color of segment
`ticker_label` (`str` or `None`): Axis label (e.g., "Q3") for filtering
`segmentation_model` (`str`): Segmentation model to use; `"color"` for color-based filtering or `"SAM"` for Segment Anything (`"color"` default)

Returns:
`tuple[int,int,int,int]`: Bounding box `(x, y, w, h)` representing the segment's radial coordinates

Example:

```
image = Image.open("radial_bar_plot.png")
radial_coords = get_radial(image, rgb_of_interest=(106, 184, 209), ticker_label="Q2")
print(radial_coords)
```

"""

def analyze_radial_geometry (image: PIL.Image.Image, contour_of_interest: np.ndarray) → tuple[PIL.Image.Image, int, int, float, float]:

""" Estimates the radial geometry of a radial bar chart for the segment of interest. Identifies the chart center, detects the outer circle representing the maximum value, and computes the maximum radial extent (i.e., radius) of the contour of interest. Commonly used for radial bar plots.

Args:
`image` (`PIL.Image.Image`): Input chart image
`contour_of_interest` (`np.ndarray`): Contour representing the segment of interest

Returns:
`PIL.Image.Image`: Image with detected outer circle and chart center marked
`int`: X-coordinate of the circle's center
`int`: Y-coordinate of the circle's center
`float`: Outer circle radius (`r_outer`)
`float`: Maximum radius from center to the contour (`r_max`)

Example:

```
image_radial_geometry, center_x, center_y, r_outer, r_max = analyze_radial_geometry(image,
     contour_of_interest=contour) #contour from get_radial()
display(image_radial_geometry)
print("Center coordinates:", center_x, center_y, "Outer circle radius:", r_outer, "Max
    radius:", r_max)
```

"""

def estimate_radial_value (image: PIL.Image.Image, center_x: int, center_y: int, r_outer: int, r_max: int, reference_circle_value: float) → float:

""" Estimates the value of a radial segment in a radial bar chart by scaling its radial length relative to the outermost circle. The reference value for the outer circle is provided externally (e.g., by an LLM), with a default of 100.

Args:
`image` (`PIL.Image.Image`): Input chart image
`center_x` (`int`): X-coordinate of the circle center
`center_y` (`int`): Y-coordinate of the circle center
`r_outer` (`int`): Radius of the outer circle
`r_max` (`int`): Maximum radius from the center to the contour
`reference_circle_value` (`float`): Value corresponding to the outer reference circle (default: `100`)

Returns:
`float`: Estimated value of the radial segment

Example:

```
    radial_value = estimate_radial_value(image, center_x=250, center_y=250, r_outer=200,
        r_max=150, reference_circle_value=100) #center_x, center_y, r_outer, r_max from
        analyze_radial_geometry()
    print("Estimated value:", radial_value)
  """
```

M.1.3 Chart Metadata Extraction

The metadata extraction prompt guides the agent to identify essential chart components, such as chart type, axis ranges, and legend entries. This metadata is then used to retrieve and condition the appropriate ICL examples, and to parameterize subsequent tool calls.

Chart Metadata Extraction Prompt

Instruction:
You are a vision-language model tasked with analyzing a data visualization chart image.
Extract and return the following information as a JSON dictionary using the exact keys specified below.

- `chart_type`: e.g., pie chart, multi-ring pie chart, bar chart, line chart, box plot, etc.
- `title`: Exact chart title as shown.
- `legend`: List or dictionary of all legend entries.
- `highlevel_legend_categories` and `finegrained_legend_subcategories`: If the chart shows category hierarchy, list both, even if names overlap.
- `legend_embedded`: `true` if legend is within the chart; `false` if outside.
- `x axis`/ `y axis`/ `right-y axis`/ `color-bar` labels: Axis labels (strings). May be empty.
- `x axis`/ `y axis`/ `right-y axis`/ `color-bar`/ `radial axis` ticker values: Tick values (List). May be empty.
- `annotation_type`: Either `"annotated"` or `"unannotated"`.
  - `"annotated"` – if numeric values are written directly in the chart.
  - `"unannotated"` – if such values are not shown in the chart.
- `visual_description`: Concise summary of the chart's visual structure.

Only output the JSON object.

Input:

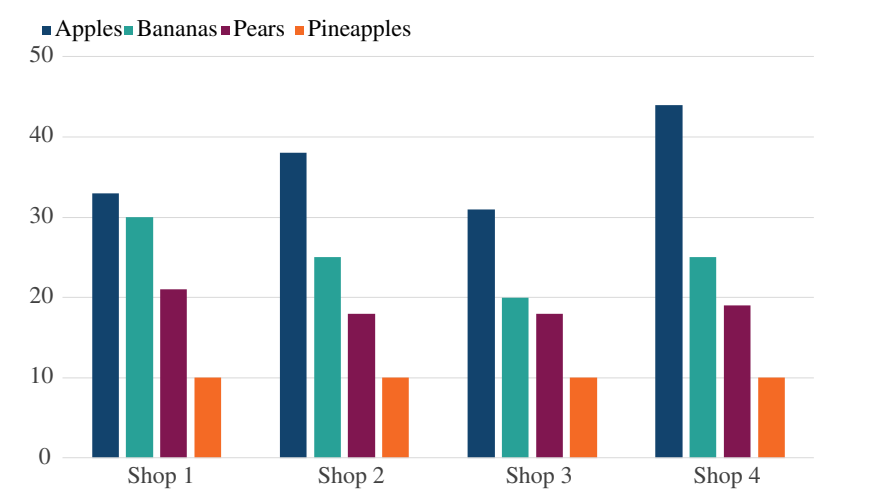


Chart image corresponding to the current sample.

M.1.4 In Context Learning

We provide ICL examples corresponding to each major chart type. At inference time, only the examples matching the detected chart type are retrieved and used. For instance, if a chart is classified as a pie chart during the metadata extraction stage, only pie chart ICL examples are appended to the prompt. If no ICL

examples exist for the detected chart type, then no ICL is added.

ICL Example Prompt

Below is an example of how to use the tools to solve a user query with a vertical grouped bar chart.

[MORE INSTRUCTIONS ...] The actual ICL prompt contains additional detailed guidelines, including self-verification, recovery strategies, changing tool parameters, using alternative tool options in case of incorrect trajectories, and fallback options to the base MLLM.

USER REQUEST #: <An image here> According to this chart, what is the GDP growth rate for India in 2020?
USER IMAGE stored in chart_image, as a PIL image.

THOUGHT 0: First, I will display and read the image.
ACTION 0:

```python
display(image=chart_image)
```

OBSERVATION: Execution success. The chart image is displayed: <image displayed here>

—

THOUGHT 1: Next, I will read the chart components such as title, legend, and axes.
ACTION 1:

```python
title_text = "Annual GDP Growth Rate"
legend_texts = {"Singapore", "India", "United States", "United Kingdom", "China"}
x_axis_tickers = ["2016", "2017", "2018", "2019", "2020"]
y_axis_tickers = ["0", "1.50", "3.0", "4.50", "6.0", "7.50", "9.0", "10.50"]
x_axis_title = "Year"
y_axis_title = "GDP Growth Rate (%)"
print(title_text, legend_texts)
```

OBSERVATION: Execution success. Title, legend, and axes are recorded.

—

THOUGHT 2: Now, I should annotate the legend in the chart image so that I can correctly identify the marker corresponding to the category of interest "India".
ACTION 2:

```python
legend_image, labeled_legend, bbox_mapping = annotate_legend(chart_image, legend_texts)
display(labeled_legend)
print (bbox_mapping)
```

OBSERVATION: Execution success. The output is a labeled image of legend elements and their bounding boxes. Labeled legend image is displayed: <image displayed here>

—

THOUGHT 3: I will now use the `labeled_legend` to determine the bounding box label corresponding to the legend marker of interest — *India*. Markers are small colored circles or squares placed to the left of their respective labels. The *India* marker is red, with the number 5 above it, so I assign 5 as the `label_of_interest`.
ACTION 3:

```python
label_of_interest = 5
print(label_of_interest)
```

OBSERVATION: Execution success. Output: `label_of_interest = 5`

—

THOUGHT 4: I will now extract the RGB color of the *India* legend marker using its bounding box label.
ACTION 4:

```python
rgb_of_interest = get_marker_rgb(legend_image, bbox_mapping, label_of_interest=5)
print ("Detected RGB color:", rgb_of_interest)
```

OBSERVATION: Execution success. Output: `Detected RGB color: (145, 20, 45)`

—

THOUGHT 5: I will now clean the chart image by removing both the title and the legend to avoid interference with bar identification using segmentation and color matching. This ensures that extraneous elements do not affect contour detection or segmentation. If the legend is too close to the x- or y-axis, I will not crop the legend to avoid removing parts

of the axis; in that case, only the title will be removed.

ACTION 5:

```python
cleaned_image = clean_chart_image(chart_image, title = title_text, legend=legend_texts)
display(cleaned_image)
```

OBSERVATION: Execution success. Cleaned image displayed without title and legend: `<image displayed here>`

—

THOUGHT 6: I will now detect the bar of interest in the `cleaned_image` corresponding to *India* (marker color = `rgb_of_interest`) and the year *2020* (`ticker_label`). The bar orientation is vertical.

ACTION 6:

```python
bar_of_interest = get_bar(cleaned_image, rgb_of_interest=rgb_of_interest, ticker_label
    ="2020", bar_orientation="vertical")
print(bar_of_interest)
```

OBSERVATION: Execution success. Output: `bar_of_interest = (120, 210, 35, 180)`

—

THOUGHT 7: Now, I will compute the value height of the detected bar (`bar_of_interest`) by mapping its pixel height to the axis scale. Since this is a vertical bar chart, I will apply a threshold of 0.2 to automatically detect the axis region in the left 20% of the image. I will then use the `x_axis_tickers` and `y_axis_tickers` (already extracted in previous steps) to perform the mapping accurately.

ACTION 7:

```python
value_height = compute_bar_height(cleaned_image, bar_of_interest, bar_orientation="
    vertical", axis_threshold=0.2, x_axis_tickers=x_axis_tickers, y_axis_tickers=
    y_axis_tickers, x_axis_title=x_axis_title, y_axis_title=y_axis_title)
print("Bar value height:", value_height)
```

OBSERVATION: Execution success. Output: `Bar value height: 4.2`

—

ANSWER: The GDP growth rate for India in 2020 is 4.2. **TERMINATE.**

—

M.2 Baseline Prompts

To benchmark `ChartAgent`, we compare against several baseline prompting strategies. We apply zero-shot (M.2.1) and chain-of-thought (CoT) (M.2.2) prompts across all proprietary and open-weight MLLM baselines. In addition, we include a ReAct prompt (M.2.3) for ablation studies, comparing `ChartAgent` with a ReAct-style agent to isolate the effect of chart-specialized visual tools. Finally, we use a tabular question-answering prompt (M.2.4) for a few chart-based baselines that output structured tables instead of direct answers.

M.2.1 Zero-shot

The zero-shot prompt provides only minimal task instructions, requiring the model to answer directly from the chart without intermediate reasoning or tool use.

Zero-shot Prompt

Instruction:
You are a data analyst skilled at analyzing chart data. Carefully examine the chart and answer the User's question with a single word or short phrase.

Input Format:

```
<chart image> {entry["image"]}
<question> {entry["query"]}
```

M.2.2 Chain-of-Thought

The chain-of-thought (CoT) prompt encourages the model to reason step by step before providing its final answer, resulting in more structured and coherent reasoning compared to zero-shot prompting.

Chain-of-Thought (CoT) Prompt

Instruction:
You are a data analyst skilled at analyzing chart data. Analyze the user's chart, carefully examine it, think step by step, and answer the user's question. Provide your final answer in the exact format: 'My final answer is {answer here}'.

Input Format:

```
<chart image> {entry["image"]}
<question> {entry["query"]}
```

M.2.3 ReAct

The ReAct prompt (Yao et al., 2023) combines reasoning traces with action steps, allowing the model to interleave thought, tool/code invocation, and observations until a final answer is reached. We use this prompt in our ablation studies to isolate the contribution of chart-specialized visual tools in our framework.

ReAct Prompt

Instruction:
You are a data analyst skilled at analyzing chart data. Your task is to analyze the provided chart and answer the user's question. Carefully examine the chart, reason step by step, and invoke actions (e.g., tool calls or code) when helpful. Follow this exact format:

```
Thought: ...
Action: ...
Observation: ...
(repeat Thought/Action/Observation as needed)

Final Answer: ...
```

If no action is needed, go directly to Final Answer.

Input Format:

```
<chart image> {entry["image"]}
<question> {entry["query"]}
```

M.2.4 Tabular Question-Answering

For a few chart-based baselines that output structured tables rather than direct answers, we apply a tabular question-answering prompt. This prompt instructs the GPT-4o model to use the extracted table together with the user's question to produce a concise answer.

Tabular Question-Answering Prompt

Instruction:
Given the data table and the user's question, use the table to determine and provide the answer in a single word or short phrase.

Input Format:

```
<extracted_table> {entry["extracted_table"]}
<user_question> {entry["query]}
```

M.3 Evaluation Prompts

Recall that we evaluate model predictions using two strategies: (1) a standardization-based accuracy computation, and (2) a GPT-Accuracy metric based on the LLM-as-a-Judge paradigm. The first method uses GPT-4o to standardize responses before applying an arithmetic-based correctness check, with a strict 5% relative error tolerance for numeric responses and string matching for non-numeric ones. The

second method prompts an LLM to assess correctness directly, also applying a 5% tolerance for numeric responses. The prompts used for both evaluation strategies are provided in M.3.1 and M.3.2, respectively.

M.3.1 Accuracy

The following prompt is used to standardize both the ground truth and predicted responses before performing the accuracy check. GPT-4o is instructed to remove units (e.g., "K" for thousand, "M" for million, "B" for billion), convert scales, eliminate symbols, and standardize number formats. Once standardized, numeric responses are evaluated arithmetically using a strict 5% relative error tolerance, while non-numeric responses require string match.

Prompt for Standardizing Ground Truth and Predicted Responses

Instruction:
You are given a question, a ground truth answer, and a model's predicted answer. Your task is to determine whether the prediction is correct.

Follow these steps exactly:

1. If both answers are **numeric**, first extract the numeric portion of each value.
   - Normalize both answers carefully by checking both the `<groundtruth answer>` and `<predicted answer>` values in context.
   - If both values include the **same unit** (e.g., `K`, `M`, `B`, or their full-word equivalents like `thousand`, `million`, `billion`), **do not scale**. Simply strip the unit and compare the base numbers.
   - Example: `20K` vs `21K` → compare `20` vs `21`.
   - If one value has a unit and the other is already scaled (e.g., `21K` vs `21000`), convert the unit-based value to its full numeric form before comparison.
   - Example: `21K` vs `21000` → compare `21000` vs `21000`.
   - If only one value includes a unit and the other is **not clearly scaled**, strip the unit and compare only the numeric parts (without scaling either).
   - Examples:
     - `16M` vs `16` → compare `16` vs `16`
     - `500M` vs `500` → compare `500` vs `500`
   - Apply scaling **only when the context requires it**, based on whether the values represent scaled vs. unscaled forms of the same quantity.
   - Additional examples:
     - `20K` vs `21K` → compare `20` vs `21`
     - `20K` vs `21000` → compare `20000` vs `21000`
     - `16M` vs `16` → compare `16` vs `16`
     - `500` vs `500M` → compare `500` vs `500`
     - `142.6` vs `1,350 million` → compare `142.6` vs `1350`
     - `170.0` vs `160 million` → compare `170` vs `160`
     - `20B` vs `21` → compare `20` vs `21`
   - Convert number words (e.g., `ten`, `eleven`, `forty-two`) to digits.
   - Remove commas, currency symbols, and surrounding text.
   - Ignore case when processing text.
2. If the answers are not numeric, return the string response.

Return a JSON object in the following format:

```
{
  "ground_truth_filtered": <normalized ground truth value>,
  "response_filtered": <normalized predicted value>
}
```

Input Format:

```
<question> {entry["query"]}
<groundtruth answer> {entry["ground_truth"]}
<response> {original_response}
```

M.3.2 LLM-as-a-Judge

The following prompt is used to evaluate response correctness using the LLM-as-a-Judge baseline, also referred to as *GPT-Accuracy* in prior literature (Xu et al., 2023; Masry et al., 2022; Xia et al., 2024). The LLM (GPT-4o) is shown the question, ground truth, and model prediction, and is asked to assess

whether the prediction is correct, with a 5% error tolerance applied to numeric answers. While flexible, this method may be imprecise for fine-grained numeric evaluation, as discussed in Sections 4 and K.5.

LLM-as-a-Judge Prompt for Evaluating Response Correctness

Instruction:
Given multiple QA pairs and the corresponding predictions, evaluate the correctness of each prediction. Return only a single word: `"True"` or `"False"`.
If the ground truth answer is a numeric value (with or without units), allow a **5% error tolerance** when comparing against the prediction.

Input Format:

```
<question> {entry["query"]}
<groundtruth answer> {entry["ground_truth"]}
<response> {original_response}
```

M.4 Complexity Analysis Prompts

Each chart–question pair in our dataset is annotated with two types of complexity labels: visual complexity and reasoning complexity. The prompts used to generate these labels are shown in M.4.1 and M.4.2, respectively.

M.4.1 Visual Complexity

The following prompt categorizes charts by visual complexity—Easy, Medium, or Hard—based solely on the visual effort needed to interpret the information presented in the chart image.

Visual Complexity Rating Prompt

Instruction:
You are a data analyst, skilled at visually interpreting charts. You are shown only a **chart image**, with no additional context or question. Your task is to assess the **visual complexity** of the chart based solely on what you see.

Think step by step:

- Count the number of visible elements (e.g., data points, lines, bars, rings, legends, labels, colors, gridlines).
- Assess how cluttered or clean the layout appears.
- Judge whether understanding the chart requires visual comparisons, dense reading, or reasoning over multiple components.
- Examine the clarity of legends, axis ticks, text placement, and grouping.

Charts tend to be visually complex if they include:

- Multiple nested or layered elements
- 3D perspectives or overlapping dimensions
- Low contrast or overlapping visual elements
- Radar/polar charts with filled or intersecting shapes
- Multi-axis designs (e.g., dual Y-axes)
- Overlaid plot types (e.g., bars + lines)
- Dense scatter plots with tightly packed points
- Stacked formats requiring segment comparison
- Ambiguous or visually similar elements that are hard to distinguish

When in doubt — especially in the presence of distortion, depth, layering, or ambiguity — prefer labeling as **Hard**.

Respond in the following JSON format:

```
{
  "label": "Easy / Medium / Hard",
  "reasoning": "Step-by-step explanation of how you reached your conclusion."
}
```

Input Format:

```
<chart image> {entry["image"]}
```

M.4.2 Reasoning Complexity

The following prompt categorizes chart–question pairs by reasoning complexity—Easy, Medium, or Hard—based solely on the level of reasoning needed to interpret and answer the question using the chart image.

Reasoning Complexity Rating Prompt

Instruction:
You are a data analyst, skilled at solving visual questions over chart images. You are shown a chart image and a corresponding question. Your task is to assess the **reasoning complexity** required to answer the question correctly.

Think step by step:

1. Identify the key visual elements referenced by the question.
2. Determine the number of distinct reasoning steps needed to answer it.
3. Evaluate the complexity of each step, considering:
    - The need for precise perception (e.g., color or shape differentiation, relative positioning)
    - Cross-referencing multiple regions, axes, or visual types
    - Complex chart features (e.g., stacked vs. overlayed areas, 3D perspective)
    - Occlusion or ambiguity in label visibility (e.g., overlapping text or hidden legends)
    - Requirement for highly precise numerical interpretation (especially in visually challenging layouts)

Typically *Hard* cases include:

- Area charts requiring distinction between stacking vs overlay
- 3D charts with visual distortion or unclear projections
- Double-ring pie charts requiring ring disambiguation
- Radar charts with overlapping regions
- Multi-axis charts with axis disambiguation needs
- Perceptually ambiguous cases (e.g., boxplots with red boxes and red medians)
- Multi-step numerical comparison questions (e.g., "How much higher is X than Y?")
- Charts with occluded or obscured labels or legends

Typically *Medium* cases include:

- Multi-bar or multi-line charts with separated groups
- Node-link diagrams requiring structured inspection
- Two-step quantitative reasoning (e.g., compute and compare X and Y)
- Tasks involving comparison or arithmetic over multiple extracted values

Typically *Easy* cases include:

- Annotated charts where answers can be read off directly
- Questions solvable via clearly readable text
- Simple selection (e.g., identifying the maximum value)

Respond in the following JSON format:

```
{
  "label": "Easy / Medium / Hard",
  "reasoning": "Step-by-step explanation of how you reached your conclusion."
}
```

Input Format:

```
<chart image> {entry["image"]}
<question> {entry["query"]}
```