Beyond Pixels: Introspective and Interactive Grounding for Visualization Agents*

Anonymous ACL submission

Abstract

Coding agents rely on Vision-Language Models (VLMs) to verify the charts they generate, but VLMs frequently hallucinate data values from rendered pixels and fail to detect fine-grained errors. We propose **Introspective and Interactive Visual Grounding (IVG)**, a framework with two complementary mechanisms: (1) **spec-grounded introspection**, which provides direct access to the agent's own chart specification for deterministic value verification; and (2) **view-grounded interaction**, which allows agents to manipulate the view (zoom, pan, toggle traces) to resolve ambiguity in overlapping regions. To evaluate this framework, we introduce **iPlotBench**, a benchmark of 500 interactive Plotly figures with 6,706 binary questions and released ground-truth specifications. Our experiments show that spec-grounded introspection improves data reconstruction fidelity in chart recreation, while view-grounded interaction yields the largest gains on questions requiring spatial reasoning over overlapping geometries. The combination achieves the highest overall QA accuracy, demonstrating that these two capabilities are complementary for building trustworthy visualization agents.

1 Introduction

Coding agents increasingly generate visualizations to support data analysis. After producing a chart, these agents often need to verify its correctness, answer user questions about it, or refine it based on feedback. State-of-the-art approaches (Yang et al.; Li et al., 2025a; Zhao et al., 2025; Seo et al., 2025; Hong et al., 2025; Li et al., 2025b) use Vision-Language Models (VLMs) to interpret rendered charts. However, this pixel-based verification suffers from a fundamental limitation: VLMs frequently hallucinate data values (e.g., reading 39.5 as 40)(Kaul et al., 2024; Jiang et al., 2024; Rohrbach et al., 2018; Li et al., 2023; Leng et al., 2024), miss fine-grained details such as line intersections in dense plots, and confuse visually similar elements like legend colors. These errors are difficult to detect and can propagate into downstream analysis.

However, when an agent has access to a chart's underlying specification, pixel-based interpretation is no longer the only option. Modern visualization libraries like Plotly produce both a rendered image and a structured specification that encodes exact data values and visual attributes. When an agent creates a chart using such a library, it gains access to this specification and can inspect it directly rather than relying on error-prone visual perception. Furthermore, interactive charts support view manipulation such as zooming, panning, and trace toggling, allowing agents to resolve ambiguity in dense or overlapping regions.

These capabilities exist but remain unexploited by current agents. We propose **Introspective and Interactive Visual Grounding (IVG)**, a framework that equips agents with two complementary mechanisms: *Spec-grounded introspection* provides read-only access to the agent's own chart specification, enabling **deterministic verification** of exact values and attributes. *View-grounded interaction* provides tools for view manipulation, enabling agents to obtain focal context when static views are ambiguous.

Evaluating such agents is challenging: existing benchmarks like ChartQA (Masry et al., 2022) and ChartMimic (Yang et al.) treat charts as static images and often rely on VLM-based scoring, which inherits the same unreliability we aim to address. To enable deterministic evaluation, we introduce **iPlotBench**, a benchmark of 500 interactive Plotly figures with 6,706 binary questions. Unlike prior benchmarks, iPlotBench releases ground-

*Code and data available at https://anonymous.4open.science/r/IVG-D433 and https://anonymous.4open.science/r/iPlotBench-D0C4

truth chart specifications, allowing direct comparison between agent-produced and reference charts without VLM-based judgment.

Our contributions are as follows:

1. **IVG Framework:** A Visualization State API exposing spec-grounded introspection and view-grounded interaction through Model Context Protocol (MCP) tools.
2. **iPlotBench:** A benchmark of 500 interactive figures with 6,706 binary questions and released ground-truth specifications. We propose Semantic Structural Similarity, a metric that evaluates chart recreation through direct JSON comparison using Hungarian matching, enabling deterministic evaluation without VLM-based judgment.
3. Our experiments on Claude 4.5 Haiku and Qwen demonstrate that spec-grounded introspection improves data reconstruction fidelity, while view-grounded interaction yields the largest gains on questions involving complex overlapping geometries. The combination achieves the highest overall QA accuracy.

2 Related Work

Chart Understanding and QA. Early benchmarks like FigureQA (Kahou et al., 2018) and DVQA (Kafle et al., 2018) established the task of Visual Question Answering (VQA) on synthetic plots. Subsequent datasets like ChartQA (Masry et al., 2022) and PlotQA (Methani et al., 2020) introduced real-world charts requiring complex reasoning. However, these approaches treat charts as static pixel arrays, forcing models to rely on visual approximations. Our work addresses the resulting Pixel Verification Bottleneck by allowing agents to inspect the underlying chart specification directly.

Chart-to-Code and Recreation. Existing work focuses on agents that generate code to render charts. ChartMimic (Yang et al.), Plot2code (Wu et al., 2025), Chart2Code53 (Niu et al., 2025) and ChartCoder (Zhao et al., 2025) evaluate agents on their ability to visually reproduce reference charts. However, these benchmarks rely on VLM-based evaluation (e.g., GPT-4V) or pixel-level metrics, which differ from true data fidelity. We introduce *Semantic Structural Similarity*, a metric that deterministically evaluates chart reconstruction by directly comparing the agent-produced Plotly JSON against the released ground-truth specification.

Tool-Integrated Coding Agents. Coding Agents (Zhang et al., 2024a; Yang et al., 2024; Guo et al.; Wang et al., 2024; Zhang et al., 2024b; Xia et al., 2024; Schick et al., 2023; Yao et al., 2022) demonstrate the power of tool use (e.g., terminals, linters) for general software engineering in the real-world repo. These agents are typically powered by frontier code LLMs such as Codex (Chen et al., 2021), Code Llama (Roziere et al., 2023), Gemini (Team et al., 2023), Claude (Anthropic, 2024) and Qwen-Coder (Hui et al., 2024). While these agents can run code, they typically lack "visual tools" to inspect their graphical outputs. Our framework bridges this gap by providing a visualization state interface for spec-grounded introspection and view-grounded interaction for visualization tasks.

3 IVG Design

IVG enables agents to reason about charts through structured access rather than pixel interpretation only. It builds on a key property of modern visualization libraries: when a library such as Plotly renders a chart, it produces not only an image but also a structured specification, typically a JSON object, that encodes data values, bindings and visual attributes(Satyanarayan et al., 2016; Zhu, 2013; Wickham, 2011; Wilkinson, 2011). This specification fully describes the chart and can be used to reconstruct the rendered image. IVG provides agents access to this specification through the Visualization State API, a set of interfaces for specification queries and view manipulation. Figure 1 illustrates the two mechanisms these interfaces support: spec-grounded introspection for accessing exact values, and view-grounded interaction for navigating visually complex regions.

Spec-grounded introspection gives agents direct access to the JSON specification underlying a chart. This specification encodes the complete chart state: data arrays, bindingsand visual attributes. Rather than estimating values from pixels, agents query the specification directly. To verify whether a bar represents the maximum, the agent extracts the data array and computes the answer symbolically. To confirm a legend entry, the agent reads the corresponding field rather than interpreting rendered text. This directness matters because pixel-based perception is inherently approximate. Two bars of similar height may be indistinguishable visually but differ meaningfully in value; the specification reveals the exact numbers. By grounding verification in struc-

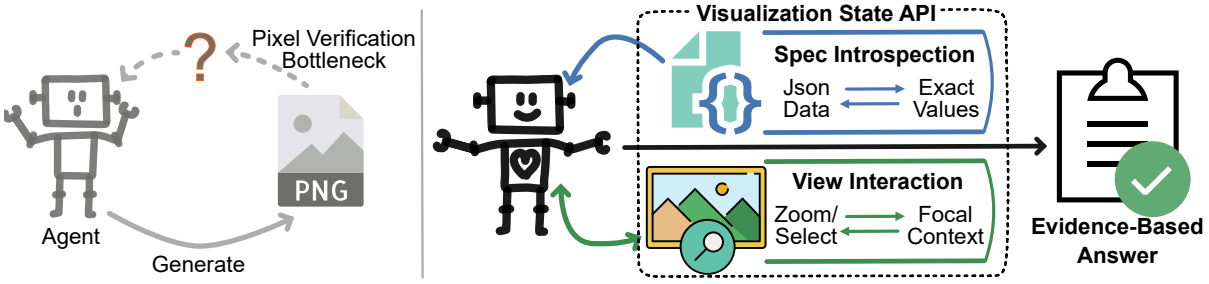

Figure 1: Overview of IVG. Rather than verifying by re-interpreting a rendered chart with a standard VLM (left), the agent uses a visualization state interface to (i) perform **spec-grounded introspection** over the chart specification for deterministic data verification and (ii) perform **view-grounded interaction** (e.g., zoom/select) to obtain focal context in visually ambiguous regions, producing an answer supported by auditable evidence.

tured data, agents avoid the estimation errors and hallucinations that arise from pixel interpretation.

While spec-grounded introspection provides exact values, it requires knowing which part of the specification to examine. A chart may contain hundreds of data points across multiple traces; querying all of them is neither efficient nor informative. View-grounded interaction addresses this challenge by allowing agents to manipulate the visual presentation to identify relevant subsets of data. When an agent zooms into a region, the adjusted axis ranges define a spatial filter over the data. When an agent toggles trace visibility, it isolates specific series for focused examination. When an agent selects a rectangular region, the system returns the data points contained within. These operations do not modify the underlying data, but they produce context that guides subsequent queries: which traces to examine, which ranges to filter, which points to inspect. This focal context bridges the gap between a high-level question and the precise specification query needed to answer it.

These two mechanisms work together in a unified workflow. Given a question about a chart, an agent first determines whether the relevant data can be directly located in the specification. If so, spec-grounded introspection suffices: the agent queries the specification and extracts the answer. If the question involves a visually complex region where the relevant data is not immediately identifiable, the agent first uses view-grounded interaction to obtain focal context, then queries the specification within that focused scope. For example, to determine whether two lines intersect, an agent might zoom into the region where they appear close, then query the specification for exact coordinates in that range. This combination ensures that answers are grounded in the specification rather than inferred from pixels. The workflow is iterative: if the gathered evidence reveals inconsistencies, the agent can refine its focus and query again until a well-supported answer emerges.

Category	Tool	Args	Function
Base	show_plot	plotly_codes	Creates figure; returns plot_id
	get_plot_image	plot_id	Returns current view as PNG
Introspection	get_plot_json	plot_id	Returns full Plotly specification
Interaction	relayout	plot_id, x/y_range	Zoom/Pan axis bounds
	legendclick	plot_id, trace_idx	Toggle trace visibility
	selected	plot_id, x/y_range	Returns points in region
	query_interactions	plot_id	Returns interaction history

Table 1: Visualization State API. *Base* tools enable plot creation and pixel capture; spec-grounded introspection provides read-only access to the agent's own Plotly specification; view-grounded interaction supports view manipulation and context queries.

The Visualization State API is implemented as a set of Model Context Protocol (MCP) tools, summarized in Table 1. These tools support figure creation, specification access, and view manipulation as described above. Throughout the workflow, agents access only specifications of figures they have created; ground-truth data and reference solutions remain hidden to ensure valid evaluation.

4 iPlotBench Dataset

Existing visualization benchmarks, such as ChartMimic (Yang et al.) and FigureQA (Kahou et al., 2018), treat charts as static images. This makes it difficult to evaluate agents that (1) use view interaction (e.g., zooming into dense regions or toggling traces) and (2) require a rendered chart specification for deterministic, structure-level evaluation, rather than pixel- or VLM-based scoring.

To bridge this gap, we introduce iPlotBench, a benchmark for evaluating **spec-grounded introspection** and **view-grounded interaction** in visualization agents. iPlotBench procedurally gen-

Type	#Figures	#Questions	Q/Fig
Line	100	1,504	15.0
Dot-Line	100	1,690	16.9
Vertical Bar	100	1,142	11.4
Horizontal Bar	100	1,187	11.9
Pie	100	1,183	11.8
Total	500	6,706	13.4

Table 2: Dataset statistics. The benchmark features a balanced distribution of questions across five chart types, with an average of 13.4 questions per figure.

erates interactive Plotly figures and releases their ground-truth specifications, enabling deterministic evaluation of chart recreation and visual reasoning.

4.1 Dataset Construction

The benchmark consists of 500 figures balanced across five chart types: Line (100), Dot-Line (100), Vertical Bar (100), Horizontal Bar (100), and Pie (100). We adapt the procedural generation pipeline from FigureQA (Kahou et al., 2018) and convert each instance into an interactive Plotly figure. All figures support standard interactions (zoom/pan, legend toggling, selection), enabling agents to actively probe ambiguous regions.

Each figure is paired with a rendered PNG input and its ground-truth Plotly JSON specification. Table 2 summarizes the dataset: 6,706 binary questions across 15 templates, with a rigorous 50/50 Yes/No balance per figure to prevent class-imbalance shortcuts.

The 15 question templates span three types: *Aggregation* (min/max/median, e.g., "Is Blue Violet the minimum?"), *Comparison* (value ordering, e.g., "Is Blue less than Red?"), and *Topology* (intersection/smoothness, e.g., "Does Blue intersect Red?").

4.2 Tasks

We define two evaluation tasks that target complementary capabilities. In our evaluation protocol, the tasks are sequential within the same agent session: the agent first recreates the chart (Task 1), then answers questions based on the recreated chart (Task 2).

Task 1: Chart Recreation. Given a static reference image, the agent must generate the underlying Plotly JSON specification that faithfully recreates the chart. Unlike previous "chart-to-code" tasks that allow arbitrary styling, this task requires precise structural grounding: mapping visual elements (e.g., bar heights, colors, labels) to exact attributes in the JSON schema. Success in this task demonstrates the agent's capability for symbolic verification—interpreting visual pixels as deterministic data structures.

Task 2: Visual QA. Given a chart and a binary question, the agent must answer yes (1) or no (0). Many questions are designed to be visually ambiguous in a static view (e.g., detecting intersections in overlapping line charts or identifying the minimum in a dense bar chart). Agents with view-grounded interaction can leverage zoom, pan, legend toggles, and region selection to resolve such ambiguities.

4.3 Semantic Structural Similarity

For Task 1, we propose Semantic Structural Similarity, which evaluates semantic correctness of agent-generated Plotly figures via direct JSON inspection. Unlike recent benchmarks such as ChartMimic (Yang et al.) which rely on VLM-based scoring (e.g., GPT-4o) to assess data trends and can be sensitive to VLM hallucinations, our metrics provide deterministic, symbolic verification. Exact JSON matching is unsuitable due to trace reordering and variable sampling density (e.g., different discretizations of the same curve). We therefore align predicted traces (P) to ground-truth traces (T) using Hungarian matching on data geometry: we compute pairwise Chamfer distances and solve the linear assignment to obtain a set of matched trace pairs $Pairs$. All trace-based metrics use the shared denominator $\max(|T|, |P|)$ to penalize extra/missing traces; unmatched traces contribute zero.

We intentionally omit separate layout and clarity metrics: in our single-plot figures, layout is largely captured by role-aware text, while clarity is subjective and is typically reflected as data/style errors.

Chart Type (S_{Type}) We verify the trace type (e.g., *scatter*, *bar*) for each matched pair:

$$S_{Type} = \frac{\sum_{(t,p)\in Pairs} \mathbb{1}[t.type = p.type]}{\max(|T|, |P|)} \quad (1)$$

Data Accuracy (S_{Data}) We treat each trace as a point cloud and compute Chamfer distance between matched pairs, robust to sampling differences. Within Chamfer, we use Euclidean distance on range-normalized numerical dimensions and a Jaccard-based penalty for categorical/text dimen-

sions (e.g., bar labels):

$$S_{Data} = \sum_{(t,p)\in Pairs} \frac{e^{-\lambda D_{Chamfer}(t,p)}}{\max(|T|,|P|)} \quad (2)$$

We set $\lambda = 5$ for range-normalized distances ($D \in [0,1]$), so a 20% deviation ($D = 0.2$) yields ≈ 0.37 and a maximum error ($D = 1.0$) yields near-zero.

Text Correctness (S_{Text}) We use a role-aware text metric to avoid "right text, wrong place" errors. Text is bucketed into semantic roles $\mathcal{R}$ (Title, Axis, Legend, Data/Annotations), and compared with a fuzzy Jaccard similarity:

$$S_{Text} = \sum_{r\in\mathcal{R}} w_r \cdot \text{Sim}(Text^r_T, Text^r_P) \quad (3)$$

Visual Style (S_{Style}) We average style similarity over matched traces and properties $\mathcal{P} = \{color, mode, symbol, size, dash, width\}$:

$$S_{Style} = \frac{\sum_{(t,p)\in Pairs} \sum_{prop\in\mathcal{P}} S_{prop}(t,p)}{\max(|T|,|P|)\,|\mathcal{P}|} \quad (4)$$

Here $S_{prop}(t,p)$ compares the values of property $prop$ in traces t and p. For color, we compare in CIELAB space (and use EMD for color arrays); categorical attributes use exact match and numerical attributes use normalized error, comparing against Plotly defaults when an attribute is omitted.

4.4 QA Evaluation

For Task 2, we report standard accuracy over all binary questions:

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[\hat{y}_i = y_i] \quad (5)$$

where $\hat{y}_i$ is the predicted answer and y_i is the ground truth.

5 Evaluation

5.1 Evaluation Setup

Configurations and Protocol. We compare four configurations: *Vision* (no tools), *+Inter* (interaction-only tools), *+Intro* (introspection-only via *get_plot_json*), and *Full* (both introspection and interaction tools). For each iPlotBench figure, the agent first recreates the chart from a static reference image (Task 1), then answers binary questions about its recreated chart within the same session (Task 2). Tools are available but tool use is optional; all configurations share the same prompt template and tool schemas (Appendix B).

Models. Our main ablation uses Claude 4.5 Haiku. We also run a model scaling study on Qwen-VL family (Table 3) for model scaling analysis. Because provider runtimes may include undisclosed system prompts, we emphasize *within-model* deltas (e.g., Vision vs Full) for causal claims.

Model	Architecture	Active Params
Qwen3-VL-30B-A3B	MoE	3B
Qwen3-VL-32B	Dense	32B
Qwen3-VL-235B-A22B	MoE	22B
Qwen-VL-Max	Proprietary	>72B (est.)

Table 3: Qwen-VL models evaluated for scaling analysis. MoE models route to a subset of parameters per token (Active Params).

Results summary. We evaluate four agent configurations on iPlotBench. Within each episode, the agent first recreates the chart from the static reference image (Task 1), then answers binary questions by reasoning over its recreated interactive figure (Task 2). Overall, spec-grounded introspection primarily improves chart reconstruction, while combining introspection with interaction yields the strongest QA performance.

5.2 Task 1: Chart Recreation

Metric	Vision	+Inter	+Intro	Full
S_{Type}	0.9542	0.9742	**0.9807**	0.9744
S_{Data}	0.8847	0.8935	**0.9016**	0.8907
S_{Text}	0.9755	**0.9903**	0.9847	0.9877
S_{Style}	0.8699	0.8708	**0.8749**	0.8679

Table 4: Semantic Structural Similarity for Task 1.

As shown in Table 4, equipping agents with introspection (*+Intro*) drives the most significant gains in semantic reconstruction, achieving the highest scores in trace typing (S_{Type}), data fidelity (S_{Data}), and style (S_{Style}). While interaction (*+Inter*) slightly outperforms in text extraction (S_{Text}), likely by exposing occluded labels, it provides limited structural benefit. Notably, the *Full* agent does not surpass *+Intro*: adding interaction expands the action space, and unnecessary view changes can distract from spec-level fixes during reconstruction.

Figure Type	Vision	+Inter	+Intro	Full
Line	0.7012	0.7030	0.7085	**0.7292**
Dot-Line	0.7155	0.7178	0.7353	**0.7522**
Vertical Bar	0.7993	0.8261	**0.8529**	0.8434
Horizontal Bar	0.8354	0.8388	**0.8623**	0.8463
Pie	0.8986	0.9019	**0.9197**	0.9146
OVERALL	0.7783	0.7850	0.8023	**0.8062**

Table 5: Question-level accuracy for Task 2.

5.3 Task 2: Visual QA

Table 5 highlights the synergy of introspection and interaction: the *Full* agent achieves the highest overall accuracy (0.8062). The benefits of interaction are most pronounced for Line and Dot-Line charts, where zooming and selective inspection help resolve overlaps. For Bar and Pie charts, *+Intro* achieves the best accuracy, suggesting that direct spec access is sufficient when the geometry is less ambiguous.

5.4 Deconfounding Reconstruction Quality

Since Task 2 is performed on the chart recreated in Task 1, QA is bounded by reconstruction fidelity. To better isolate verification and reasoning, we report Conditional QA Accuracy on the subset of figures where reconstruction quality is high ($S_{Data} \geq 0.9$).

Config	Per-figure Acc	Cond. Acc ($S_{Data} \geq 0.9$)
Vision	0.7941	0.8321
+Inter	0.8013	0.8538
+Intro	0.8190	0.8606
Full	**0.8199**	**0.8637**

Table 6: QA Accuracy conditioned on high-fidelity reconstruction ($S_{Data} \geq 0.9$).

Table 6 reports *per-figure* QA accuracy (averaging accuracy within each figure, then averaging across figures), to avoid overweighting figure types with more questions. Conditioning on high-fidelity reconstructions ($S_{Data} \geq 0.9$) increases accuracy across all configurations, with *Full* achieving the highest conditional accuracy (0.8637), indicating that IVG provides additional benefit beyond improved reconstruction.

Table 7 reveals complementary strengths: *+Intro* excels at Comparison questions where direct spec access enables precise value retrieval, while *Full* performs best on Topology questions where interaction helps resolve spatial relationships.

Config	Aggregation	Comparison	Topology
Vision	0.8354	0.8391	0.7378
+Inter	0.8591	0.8522	0.8043
+Intro	0.8846	**0.8555**	0.7567
Full	**0.8864**	0.8514	**0.8052**

Table 7: Conditional QA accuracy by question family ($S_{Data} \geq 0.9$). Aggregation: min/max/median; Comparison: less/greater; Topology: intersection, smoothness, AUC.

5.5 Model Scaling Analysis

We examine how IVG benefits scale with model capability using the Qwen-VL family.

Main Results. Table 8 presents Task 1 (S_{Data}) and Task 2 (QA accuracy) for Vision and Full configurations.

Model	S_{Data} (V)	S_{Data} (F)	QA (V)	QA (F)
Qwen3-VL-30B-A3B	0.852	0.642	0.948	0.870
Qwen3-VL-32B	0.881	**0.883**	0.956	0.933
Qwen3-VL-235B-A22B	0.878	**0.904**	0.951	**0.960**
Qwen-VL-Max	0.916	**0.922**	0.971	**0.973**

Table 8: Task 1 and Task 2 results for Qwen-VL models. V=Vision (no tools), F=Full (IVG tools). Bold indicates improvement with tools.

We observe that IVG benefits scale with model capability. Models with larger active parameters (Qwen3-VL-235B-A22B, Qwen-VL-Max) show consistent improvements, while smaller models face challenges in effectively orchestrating tool use alongside their primary reasoning tasks.

Conditional Analysis. To isolate verification capability from reconstruction quality, Table 9 reports QA accuracy conditioned on successful reconstruction ($S_{Data} \geq 0.9$).

Model	Cond. (V)	Cond. (F)	Δ
Qwen3-VL-30B-A3B	0.959	0.900	-0.059
Qwen3-VL-32B	0.973	0.964	-0.009
Qwen3-VL-235B-A22B	0.961	**0.966**	$+0.005$
Qwen-VL-Max	0.983	**0.985**	$+0.002$

Table 9: Conditional QA accuracy ($S_{Data} \geq 0.9$) for Qwen-VL models.

Larger models (Qwen3-VL-235B-A22B, Qwen-VL-Max) show consistent gains even when controlling for reconstruction quality, with Qwen-VL-Max approaching ceiling performance. The smaller Qwen3-VL-30B-A3B model does not benefit from tools in this setting, suggesting that effective tool orchestration requires sufficient model capacity.

Question Family Breakdown. Table 10 reveals which question types benefit most from IVG across model scales.

Model	ΔAgg	ΔComp	ΔTopo
Qwen3-VL-30B-A3B	−0.002	−0.094	−0.032
Qwen3-VL-32B	−0.003	−0.001	−0.074
Qwen3-VL-235B-A22B	+0.001	+0.007	+0.008
Qwen-VL-Max	0.000	+0.003	+0.008

Table 10: Conditional QA delta by question family ($S_{Data} \geq 0.9$). Agg=Aggregation, Comp=Comparison, Topo=Topology. Δ denotes Full − Vision.

Capable models (Qwen3-VL-235B-A22B, Qwen-VL-Max) show their largest gains on Topology questions (+0.008), where view-grounded interaction (zoom, selection) helps resolve spatial relationships. This aligns with IVG's design goal of providing focal context for visually ambiguous regions.

Effective use of IVG requires models with sufficient reasoning capacity to orchestrate tool calls alongside their primary tasks. In our evaluation, models with approximately 20B or more active parameters benefit consistently from IVG. This aligns with broader findings in tool-augmented LLMs, where tool use adds cognitive overhead that smaller models may struggle to manage. For frontier models approaching ceiling performance, IVG's primary value shifts from accuracy improvement to auditability, providing explicit and traceable evidence for each reasoning step.

6 Qualitative Analysis: Deep Plot

To demonstrate the practical utility of IVG, we deploy the framework in a verifiable data reporting system, **Deep Plot**. The Deep Plot system provides a multi-modal interface for automated data storytelling. Figure 2 illustrates the interface, featuring a dual-panel layout: a left panel containing an agent-generated data report (summary and insights) and a right panel hosting interactive Plotly figures in tabbed views.

Deep Plot operationalizes IVG as an **evidence-grounded reporting loop**. The agent iteratively proposes a claim, generates plots and summary statistics, and then refines the claim based on evidence it can explicitly retrieve from its visualization state.

The core capability is **view-grounded interaction as focal context**. Each insight links to a supporting plot. As a user explores the plot (zoom/pan, legend toggles, selection), the system records interaction events. When a follow-up question is asked, the agent uses *query_interactions* to retrieve the latest interaction state (e.g., axis ranges, trace visibility, selection bounds) and combines it with **spec-grounded introspection** via *get_plot_json* to retrieve exact values and attributes. This enables answers grounded in explicit evidence (e.g., "within the selected range, the maximum occurs at $x = ...$ with $y = ...$") rather than pixel-inferred guesses. In this setting, "evidence" is machine-checkable: tool-returned state (spec JSON, interaction events, selections) and computed numbers derived from that state.

7 Discussion

7.1 Beyond the Pixel Verification Bottleneck

Our findings indicate that the Pixel Verification Bottleneck is a primary source of unreliability in current visualization agents. The gains in S_{Data} from spec-grounded introspection show that giving agents access to their own chart specification enables verification against exact values and attributes, rather than probabilistic pixel inference. This directly addresses the Trust Gap identified in the introduction: important parts of visualization verification become auditable at the level of agent-owned state.

7.2 Tool Synergy and Focal Context

A critical insight from our study is the synergy between spec-grounded introspection and view-grounded interaction. While spec-grounded introspection provides exact, spec-level access to the values encoded in the agent-generated chart, it can also overwhelm an agent with irrelevant structural details. As seen in the Deep Plot system (Section 6), view-grounded interaction provides the focal context. Zooming, toggling traces, and selection update only the view state and emit interaction context (e.g., ranges and events) that localizes *which subset* of the visualization state is relevant, without changing the underlying data. This mimics the human analyst's workflow: using view-grounded interaction to focus attention and structured inspection to verify values. However, we also observed that unfettered view-grounded interaction can introduce exploration overhead. As noted in the Task 1 results, the *Full* agent occasionally underperformed due to excessive tool usage, highlighting the need for agents to strategically balance

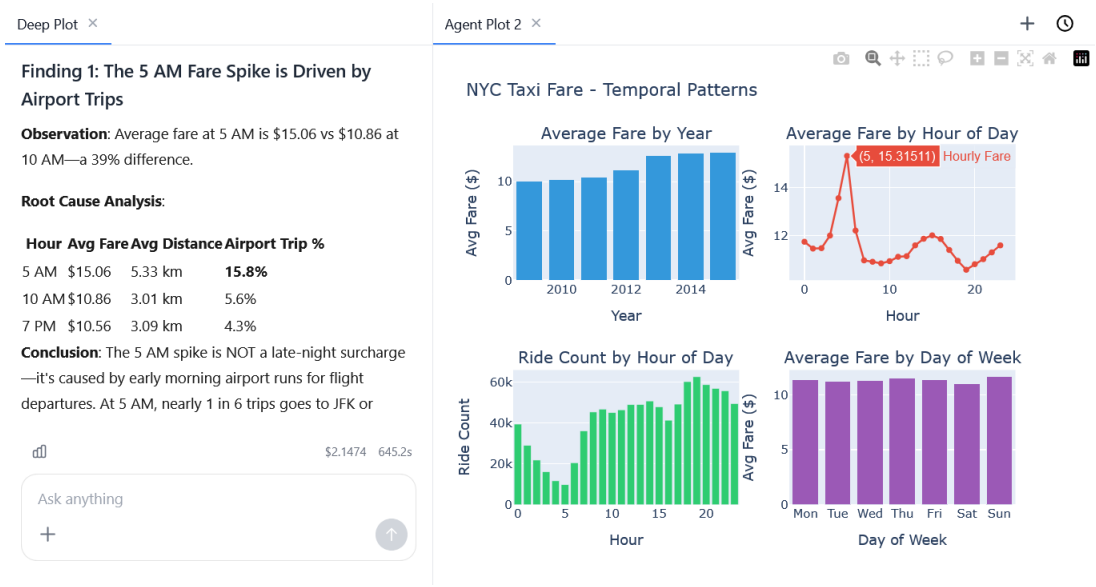

Figure 2: **Deep Plot Interface.** Insights in the report (left) reference interactive plots (right). The agent uses spec-grounded introspection and view-grounded interaction history to answer follow-up questions grounded in the user's current visual context.

between spec-grounded introspection (reading) and view-grounded interaction (acting).

7.3 Implications for Real-World Systems

Deep Plot (§6) suggests that spec-grounded introspection and view-grounded interaction are not merely diagnostic aids, but can serve as core components of user-facing analysis workflows. When follow-up questions are grounded in the current interaction state (e.g., axis ranges and selections) and verified against the underlying visualization specification, agents can respond with explicit, checkable evidence rather than pixel-only guesses. More broadly, this supports the view that visualization agents should be grounded in agent-owned visualization state to remain reliable in real-world data analysis settings.

8 Limitations

We explicitly do not claim that the current toolset is a universal solution for all visualization frameworks. While our framework is architected to be library-agnostic, our empirical validation is restricted to the Plotly ecosystem. Extending this "DOM-like" state access to other libraries (e.g., Matplotlib, D3.js) or domains (e.g., CAD, UI design) remains an engineering challenge. Second, our S_{Data} metric for pie charts currently relies on Cartesian Chamfer distance, which may not perfectly capture radial proportions; future work will explore polar-coordinate aware metrics.

Our Qwen-VL scaling study (§5.5) suggests a capability dependence in tool use: the smallest active-parameter model (Qwen3-VL-30B-A3B) degrades when given IVG tools, while larger models show more consistent benefits (with Qwen3-VL-32B exhibiting mixed effects across tasks). This points to a practical *tool capability threshold*—below which the overhead of tool orchestration can outweigh verification gains. For frontier models approaching ceiling performance (e.g., Qwen-VL-Max), IVG's value also shifts toward *auditability*: producing answers that can be traced to explicit evidence from the visualization state.

9 Conclusion

We presented Introspective and Interactive Visual Grounding (IVG), a framework that transforms agents from passive observers into active owners of their visualizations. By bypassing the Pixel Verification Bottleneck and enabling spec-grounded introspection and view-grounded interaction, we provide a path toward closing the Trust Gap in autonomous data analysis. Our results on iPlotBench demonstrate that spec-grounded introspection and view-grounded interaction are complementary: one provides deterministic evidence, while the other provides focal context. Through Deep Plot (§6), we illustrate how IVG supports evidence-grounded reporting and grounded follow-up QA in a user-facing workflow.

References

Anthropic. 2024. The claude 3 model family: A new standard for intelligence. https://www.anthropic.com/news/claude-3-family.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Jinyao Guo, Chengpeng Wang, Xiangzhe Xu, Zian Su, and Xiangyu Zhang. Repoaudit: An autonomous llm-agent for repository-level code auditing. In *Forty-second International Conference on Machine Learning*.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, and 9 others. 2025. Data interpreter: An LLM agent for data science. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19796–19821, Vienna, Austria. Association for Computational Linguistics.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Chaoya Jiang, Haiyang Xu, Mengfan Dong, Jiaxing Chen, Wei Ye, Ming Yan, Qinghao Ye, Ji Zhang, Fei Huang, and Shikun Zhang. 2024. Hallucination augmented contrastive learning for multimodal large language model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27036–27046.

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, Adam Atkinson, Vincent Michalski, Ákos Kádár, Adam Trischler, and Yoshua Bengio. 2018. Figureqa: An annotated figure dataset for visual reasoning. In *International Conference on Learning Representations (ICLR) Workshop*.

Prannay Kaul, Zhizhong Li, Hao Yang, Yonatan Dukler, Ashwin Swaminathan, CJ Taylor, and Stefano Soatto. 2024. Throne: An object-based hallucination benchmark for the free-form generations of large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27228–27238.

Sicong Leng, Hang Zhang, Guanzheng Chen, Xin Li, Shijian Lu, Chunyan Miao, and Lidong Bing. 2024. Mitigating object hallucinations in large vision-language models through visual contrastive decoding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13872–13882.

Bingxuan Li, Yiwei Wang, Jiuxiang Gu, Kai-Wei Chang, and Nanyun Peng. 2025a. Metal: A multi-agent framework for chart generation with test-time scaling. *Preprint*, arXiv:2502.17651.

Bingxuan Li, Yiwei Wang, Jiuxiang Gu, Kai-Wei Chang, and Nanyun Peng. 2025b. Metal: A multi-agent framework for chart generation with test-time scaling. *CoRR*.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Evaluating object hallucination in large vision-language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 292–305.

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.

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.

Tianhao Niu, Yiming Cui, Baoxin Wang, Xiao Xu, Xin Yao, Qingfu Zhu, Dayong Wu, Shijin Wang, and Wanxiang Che. 2025. Chart2code53: A large-scale diverse and complex dataset for enhancing chart-to-code generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 15839–15855.

Anna Rohrbach, Lisa Anne Hendricks, Kaylee Burns, Trevor Darrell, and Kate Saenko. 2018. Object hallucination in image captioning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4035–4045.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350.

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.

Wonduk Seo, Seungyong Lee, Daye Kang, Hyunjin An, Zonghao Yuan, and Seunghyun Lee. 2025. Automated visualization code synthesis via multi-path reasoning and feedback-driven optimization. *Preprint*, arXiv:2502.11140.

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*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, and 1 others. 2024. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

Hadley Wickham. 2011. ggplot2. *Wiley interdisciplinary reviews: computational statistics*, 3(2):180–185.

Leland Wilkinson. 2011. The grammar of graphics. In *Handbook of computational statistics: Concepts and methods*, pages 375–414. Springer.

Chengyue Wu, Zhixuan Liang, Yixiao Ge, Qiushan Guo, Zeyu Lu, Jiahao Wang, Ying Shan, and Ping Luo. 2025. Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3006–3028.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying llm-based software engineering agents. *CoRR*.

Cheng Yang, Chufan Shi, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran XU, Xinyu Zhu, Siheng Li, Yuxiang Zhang, and 1 others. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. In *The Thirteenth International Conference on Learning Representations*.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2024a. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13643–13658.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024b. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1592–1604.

Xuanle Zhao, Xianzhen Luo, Qi Shi, Chi Chen, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2025. Chartcoder: Advancing multimodal large language model for chart-to-code generation. *arXiv preprint arXiv:2501.06598*.

Nick Qi Zhu. 2013. *Data visualization with D3. js cookbook*. Packt Publishing Ltd.

A Visualization State API: MCP Tool Schemas

The Visualization State API is implemented as a set of Model Context Protocol (MCP) tools. MCP provides a standardized interface for language model agents to invoke external functions with typed arguments and structured JSON responses. All tools return JSON-formatted output for deterministic parsing.

A.1 Base Tools

show_plot Creates a Plotly figure from agent-generated Python code and registers it for subsequent operations.

- **Arguments:**
  - `plotly_codes` (string, required): Python code defining a variable `fig` as a Plotly figure object. Must include necessary imports.
- **Returns:** `{plot_id: int}` – A unique identifier for referencing this figure in subsequent tool calls.

get_plot_image Captures the current rendered view of a figure as a PNG screenshot.

- **Arguments:**
  - `plot_id` (integer, required): Plot identifier from `show_plot`.
  - `interaction_id` (integer, optional): Specific interaction snapshot ID. If omitted, returns the latest view state.

- **Returns:** `{image_path: str}` – Absolute path to the PNG file, which can be passed to the VLM for pixel-based inspection.

A.2 Spec-Grounded Introspection

get_plot_json Retrieves the full Plotly specification of the agent's constructed figure, enabling deterministic verification of data values and visual encodings.

- **Arguments:**
  - `plot_id` (integer, required): Plot identifier from `show_plot`.
- **Returns:** `{data: [...], layout: {...}}` – The complete Plotly JSON specification, where `data` contains trace objects and `layout` contains axis and styling configurations.

Trust Boundary: This tool exposes *only* the agent's own constructed figure specification—not the ground-truth dataset or reference solution.

A.3 View-Grounded Interaction

relayout Programmatically zooms or pans the view by setting axis ranges, providing focal context for dense or occluded regions.

- **Arguments:**
  - `plot_id` (integer, required): Plot identifier.
  - `x_min`, `x_max` (number, optional): X-axis range bounds.
  - `y_min`, `y_max` (number, optional): Y-axis range bounds.
- **Returns:** Success status.
- **Event Payload:** Records axis ranges, e.g., `{"xaxis.range[0]": 1981.97, "xaxis.range[1]": 2001.99, "yaxis.range[0]": 70.98, "yaxis.range[1]": 73.81}`.

legendclick Toggles the visibility of a specific trace, enabling isolation of overlapping series.

- **Arguments:**
  - `plot_id` (integer, required): Plot identifier.
  - `curve_number` (integer, required): Zero-indexed trace index.
- **Returns:** Success status.
- **Event Payload:** Records `{curve_number: int, expanded_index: int}`. Visibility changes are captured in the associated `plotly_restyle` event as `{update: [{visible: [...]}, [trace_indices]]}`.

selected Performs a box selection over a specified region and returns information about selected data points.

- **Arguments:**
  - `plot_id` (integer, required): Plot identifier.
  - `x_min`, `x_max` (number, optional): X-axis selection bounds.
  - `y_min`, `y_max` (number, optional): Y-axis selection bounds.
- **Returns:** Selection information.
- **Event Payload:** Records `{point_count: int, range: {x: [min, max], y: [min, max]}}`.

query_interactions Retrieves the interaction history for a figure, enabling the agent to reason about prior view manipulations.

- **Arguments:**
  - `plot_id` (integer, required): Plot identifier.
  - `event_type` (string, optional): Filter by event type (`init`, `relayout`, `legendclick`, `selected`).
- **Returns:** `{events: [{id, event_type, payload, has_screenshot}, ...]}` – Chronological list of interaction events, each with:
  - `id`: Unique interaction ID (can be passed to `get_plot_image` to retrieve historical snapshots).
  - `event_type`: One of `init`, `relayout`, `legendclick`, `selected`.
  - `payload`: Event-specific details (e.g., axis ranges for `relayout`, visibility state for `legendclick`).
  - `has_screenshot`: Whether a screenshot was captured for this interaction state.

B Controller and Evaluation Protocol

This section details the prompt templates, stop conditions, and execution environment used in our evaluation.

B.1 Prompt Templates

All agent configurations share identical prompt templates. We use minimal prompts to avoid biasing tool usage patterns.

Task 1: Chart Recreation The agent receives the reference image (base64-encoded PNG) along with the following text prompt:

```
Read ./input.png and recreate this plot.

Output the Plotly figure as JSON with
"data" and "layout" keys:
{"data": [...], "layout": {...}}
```

For configurations with tools enabled, the agent may invoke Visualization State API tools before producing the final JSON output. For the Vision baseline, the agent directly outputs JSON without tool access.

Task 2: Visual QA For each binary question, the agent receives:

```
<question text>

Reply with ONLY a single digit: 0 or 1
```

Task 2 questions are answered within the same session as Task 1, meaning the agent retains context from chart recreation and can reference its constructed figure state.

B.2 Stop Conditions

Each *task invocation* is bounded by two termination conditions:

- **Tool round limit:** Maximum 5 tool-calling rounds per invocation. If the model continues requesting tool calls after 5 rounds, the invocation terminates with partial results.
- **API timeout:** 120 seconds per API call. If the model does not respond within this window, the invocation is marked as failed.

Invocation structure: An episode consists of multiple invocations within a persistent session. Task 1 (chart recreation) is a single invocation. Task 2 questions are each processed as *separate invocations*, meaning each question receives its own 5-round tool budget. Critically, the session context (conversation history and figure state) persists across all invocations, allowing the agent to reference its Task 1 reconstruction when answering Task 2 questions.

An invocation terminates successfully when at least one valid answer is extracted: a Plotly JSON object with non-empty `data` array for Task 1, or a parseable 0/1 response for Task 2.

B.3 Execution Environment

Claude Agents (Main Ablation). Claude 4.5 Haiku agents run in isolated Docker containers based on `python:3.10-slim`. Each container includes:

- Python 3.10 with Plotly for figure generation
- Kaleido for server-side PNG rendering
- MCP server implementing the Visualization State API
- Isolated filesystem with read-only access to reference images

The containerized environment ensures reproducibility and prevents agents from accessing ground-truth data or other test cases.

Qwen-VL Agents (Scaling Study). Qwen-VL models are accessed via OpenAI-compatible APIs (local vLLM or OpenRouter). The same MCP tools are available, with images passed as base64-encoded PNGs in the `image_url` message format. Key differences from the Claude setup:

- No Docker isolation (API-based execution)
- Session state managed by Python runner script
- Same prompt templates and tool schemas
- Vendor runtimes may include undisclosed system prompts

To account for potential confounds from vendor-specific behaviors, we emphasize within-model comparisons (Vision vs Full) rather than cross-model rankings.

B.4 Tool Schema Availability

In all tool-enabled configurations (+Inter, +Intro, Full), the complete tool schema is provided to the model at session start. Tools are *available* but their use is *optional*—models autonomously decide when and whether to invoke tools based on task requirements. This design isolates model capability from prompt engineering that might bias tool usage.