VACP: Visual Analytics Context Protocol

Tobias Stähle, Péter Ferenc Gyarmati, Thilo Spinner,
Rita Sevastjanova, Dominik Moritz, Mennatallah El-Assady

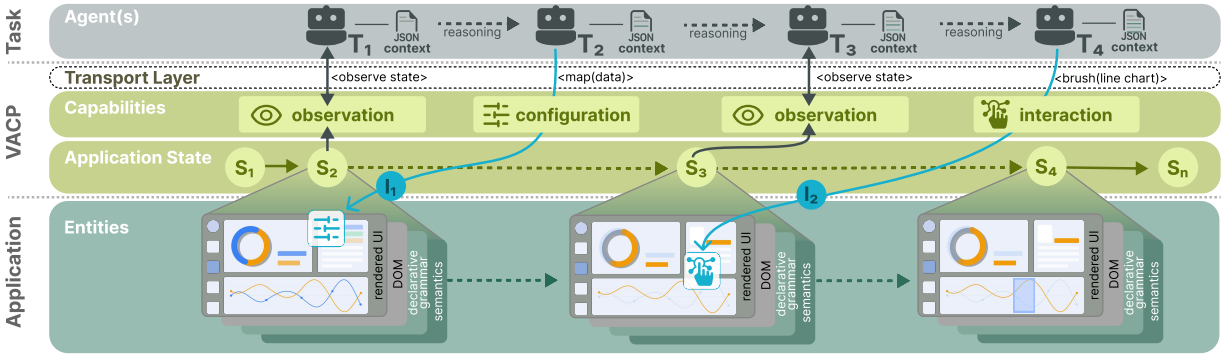

Fig. 1: AI agents interacting with Visual Analytics (VA) interfaces using **VACP** (Visual Analytics Context Protocol). VACP maintains the **application state** and available **interactions** informed through the different types of knowledge representation of the **VA entities**. To perform analytical tasks in the interface, an **AI agent** can use the provided capabilities to perceive the VA interface and understand possible interactions by using VACP. Further, it can interact with the VA interface through the provided interactions. Interactions (e.g., I_1 or I_2) manipulate the interface appearance, triggering a state transition (e.g., $S_2 \rightarrow S_3$) in the application within the VACP layer.

Abstract—The rise of AI agents introduces a fundamental shift in Visual Analytics (VA), in which agents act as a new user group. Current agentic approaches—based on computer vision and raw DOM access—fail to perform VA tasks accurately and efficiently. This paper introduces the Visual Analytics Context Protocol (VACP), a framework designed to make VA applications "*agent-ready*" that extends generic protocols by explicitly exposing application state, available interactions, and mechanisms for direct execution. To support our context protocol, we contribute a formal specification of AI agent requirements and knowledge representations in VA interfaces. We instantiate VACP as a library compatible with major visualization grammars and web frameworks, enabling augmentation of existing systems and the development of new ones. Our evaluation across a compelling set of VA tasks demonstrates that VACP-enabled agents achieve higher success rates in interface interpretation and execution compared to current agentic approaches, while reducing token consumption and completion time. VACP closes the gap between human-centric VA interfaces and machine perceivability, ensuring agents can reliably act as collaborative users in VA systems. More information under https://vacp.dev.

Index Terms—Visual Analytics, Intelligent Agents, AI Agents, Context Protocol, Agent-Ready Interfaces

1 Introduction

Visual Analytics (VA) systems are traditionally designed with the implicit assumption that ***human users*** perceive visually encoded data and cognitively process it to observe patterns. To enhance analytical workflows, mixed-initiative VA systems use machine learning models and agents to automate subtasks or provide guidance [11, 61]. These agents mainly serve as an additional component of VA systems.

Unlike traditional integrated models, recent advances in agentic AI unlock a new paradigm: ***AI users*** — autonomous agents directly operating interactive web interfaces [65], e.g., autonomously performing selection, filtering, and drill-down operations [6] to answer analytical questions [33]. In such contexts, the success of AI agents is contingent on their ability to operate and interact directly with the visual interface. If realized, VA interfaces supporting human and AI agents side by side can unlock powerful potential, enabling real-time collaboration in shared, dynamic analytical environments.

Despite this potential, current agentic approaches, such as AI agents powered by recent multimodal foundation models like Gemini [20], GPT-5.2 [45], and Claude Sonnet 4.5 [3], attempt to navigate interfaces visually using Computer Vision or raw HTML parsing [2, 9, 44, 65]. By treating agents as human simulators, these approaches *struggle to interact with VA applications*. They suffer from poor information retention [33] and frequently fail to execute interactions correctly. In early evaluations testing agents on real-world analytical tasks (Sec. 6), we observed low task completion rates driven by varied failure modes, imprecise data selection, and significant interaction execution errors.

We identify this difficulty as a critical "***interface mismatch***" that currently impedes progress. Existing VA systems are optimized for human perception (rendered pixels) and human action (mouse events), creating an environment that is hostile to AI agents. We argue that relying on pixel-level interpretation for logical tasks is flawed. It introduces unnecessary latency, stochasticity, and hallucination, forcing a high-intelligence model to operate through a low-fidelity "*keyhole*."

Breaking free from this bottleneck unlocks new possibilities for operating mixed-initiative systems. By bypassing visual approximations and exposing underlying logic and data states, we establish a direct connection between VA system and AI agent, allowing seamless retrofitting of legacy applications to support agentic accessibility tools and enabling human and AI agents to collaborate within a shared analytical context. Furthermore, the machine-readable interface serves as basis for rich programmatic evaluation environments, where researchers can systematically simulate, test, and benchmark interactive workflows.

Drawing on existing VA literature and recent work on intelligent agents in VA [40, 62], we identify that agents require access to information about the application's current state, available interaction options,

• *Tobias Stähle, Péter Ferenc Gyarmati, Thilo Spinner, Rita Sevastjanova and Mennatallah El-Assady are with ETH Zürich.*
• *Dominik Moritz is with Carnegie Mellon University.*

Manuscript received xx xxx. 202x; accepted xx xxx. 202x. Date of Publication xx xxx. 202x; date of current version xx xxx. 202x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.202x.xxxxxxx

and the ability to execute interactions. Further, we investigate which interface layers encode this knowledge to model application state and interaction representation in a way that balances semantic richness with token efficiency for the AI agent while considering limitations of current AI agents such as context window size [28, 70].

Recent general-purpose standards, such as the Model Context Protocol (MCP) [4], expose tools to AI agents but lack the domain-specific semantic grounding needed for complex, stateful VA. This paper introduces the **Visual Analytics Context Protocol** (VACP), which makes current and future VA systems "*AI-agent-ready*." VACP uses MCP as the communication substrate, and operates above its transport layers, augmenting agent context by exposing the hierarchical data structures and precise interaction spaces unique to VA systems. By exposing a structured application state and clear execution endpoints, VACP transforms VA interfaces into an environment where AI agents can perform analytical tasks reliably and accurately. This allows developers to retrofit existing systems and build new agent-ready VA systems. To our knowledge, *VACP is the first context protocol to establish a direct interaction bridge between the VA interface and AI agents as a new synthetic user type.* Specifically, we contribute:

(1) Specification of Challenges for AI Agents' Accessibility in VA: A specification of the knowledge representation non-human actors require to navigate VA systems, combining established interaction taxonomies [8, 17, 27, 69] with the different levels of knowledge representation in VA applications [60].

(2) Visual Analytics Context Protocol: A definition of principles for context protocols in interactive, data-rich environments. Based on these, we formally describe VACP, which exposes a context-rich application state, available interactions, and mechanisms for direct execution.

(3) VACP Library and Evaluation: An instantiation of VACP as a TypeScript library, supporting modern web frameworks and ensuring built-in default compatibility for higher-level declarative formats. Our evaluation compares VACP-enabled agents with state-of-the-art methods and shows that VACP agents achieve higher task completion rates while significantly reducing token consumption and execution time.

By replacing ad-hoc, heuristic observation with robust, formal communication, VACP lays the foundation for turning AI agents from passive observers into reliable analytical collaborators. Released as open-source software[1], VA developers can use VACP to *build new or supercharge existing VA systems with interfaces designed from the ground up for AI agent accessibility*.

2 Background & Related Work

Web Agents and Browser Automation — As LLMs evolved from passive text processors into active agents, research on autonomous web navigation accelerated [65]. Early automation relied on rigid scripts such as Selenium [56] or on reinforcement learning in simplified environments like *World of Bits* [58]. Instead, recent agentic approaches exploit LLMs' semantic reasoning over the Document Object Model (DOM) [67] or screenshots to execute tasks dynamically [24]. Chrome's recently proposed WebMCP [64] embeds a JavaScript interface that exposes application functionality as tools for AI browser agents to invoke. However, these approaches suffer from a critical *Interface Mismatch* in VA. DOM-based agents struggle because modern VA applications rely on `<canvas>` or complex SVG that exceeds LLM context windows [24]. To a standard DOM parser, a D3.js scatterplot is a *semantic black box*: thousands of anonymous path elements or a single opaque pixel buffer, lacking the descriptors needed for reasoning. Overcoming the context-window bottleneck would require aggressive DOM pruning, which strips the interface logic needed to understand it properly. Vision-based agents (e.g., *pixels-to-actions* [57] and computer-use agents [2, 44]) better mimic human perception but lack the coordinate precision needed in dense data environments. As Kartha et al. [33] note, this yields high hallucination rates with agents misinterpreting visual overlaps or failing to distinguish subtle color encodings. Work on visualization literacy evaluates how well multimodal LLMs interpret visualizations to assist users [12], including under varying color, shape, and text [63]. These studies find that current LLMs handle overall chart comprehension [36] and detect chart hierarchies and data trends [46] well, yet struggle with data retrieval and anomaly detection in dense visualizations [7, 46]. While benchmarks on visualization literacy and general web tasks like WebArena [72] and Mind2Web [14] exist, to our knowledge none target the interactive VA domain.

Visualization Accessibility for Agent Perception — We posit that exposing VA systems to AI agents parallels making visualizations accessible to vision-impaired users. In both, consumers cannot rely on visual gestalt and need a structured, traversable alternative [18, 32], while AI agents additionally require explicit semantic representations. Visualization accessibility has matured from simple alt-text generation to rich semantic structures. Zong et al. [73] and Lundgard and Satyanarayan [38] argue that true accessibility requires exposing not only statistical facts but also the hierarchical and semantic relationships among data, encodings, and annotations.

In VA, Sacha et al. [51] note that resolution and clutter create uncertainty for users — uncertainty that, for agents, is exacerbated by the stochastic nature of vision-language models. Declarative grammars like Vega [53], Vega-Lite [52] and Mosaic's vgplot [25, 26] provide a baseline here, as they explicitly map data to visual channels. Das et al. [13] confirm that multimodal LLMs interpret visualizations better when given such structured data. Other approaches pair LLMs with the underlying data and visualization specifications to enable natural language interfaces [31], such as VizAbility [23].

A key distinction remains: existing accessibility layers are designed for *consumption* — reading the visualizations — and rarely expose the interface *affordances* for *operation* (e.g., that brushing interval *A* filters view *B*). Kim et al. [34] stress the need for non-visual interaction modalities as interactive visualizations grow more complex, and the absence of this meta-information leads to what Gillmann et al. [21] call usage uncertainty. By extending VA's read-only semantic schemas into a read-write protocol, VACP lets AI agents actively manipulate system state, treating the agent as a user with specific "*accessibility*" needs.

Interactions in Visualizations — The importance of interaction for effective VA applications is widely recognized [47, 69] , and interaction provenance has been analyzed to study user behavior [43]. Existing taxonomies categorize interactions by user intent [69] and interactive dynamics [27], providing the vocabulary to describe what an agent intends to do. While interaction-design guidelines exist [5, 55], implementation is often tightly coupled to the rendering engine [71].

Dimara and Perin [17] argue for distinguishing an interaction's **intent** from its **physical mechanism**, decoupling utility from implementation, and enabling evaluation of techniques independent of input modalities. For mixed-initiative VA [11], where agents proactively support users, this requires a formal definition of the interaction space. Declarative grammars like Vega-Lite model the *intent* of interactions internally (via Signals and Selections), but many lack an **Interaction Gateway**, i.e., a semantic Application Programming Interface (API) that lets external agents discover and trigger intents directly without passing through the UI layer. Since the physical mechanism (e.g., simulating mouse drags) is a hindrance for current AI agents [72], recent work on agentic tool use suggests they are more effective at the intent level (API/tool calls) [54]. Advancements such as the Model Context Protocol (MCP) [4] lower this bound by exposing predefined tools in a structured format and tools like Embedding Atlas [50] already use MCP. Yet these approaches could benefit from a more efficient protocol.

We conceptualize agent-VA interactions in the UI as two stages. Stage 1 covers operating predefined interfaces (e.g., selecting, filtering, highlighting) to generate insights within fixed constraints, whereas Stage 2 involves authoring or reconfiguring interactive dashboards. Although dashboard generation and composition are popular research topics [16], an agent's ability to (re)configure a dashboard has limited value without subsequent interaction to refine findings. This paper focuses on Stage 1 interactions as the operational foundation for future complex authoring tasks, with a Stage 2 use case provided in the supplemental material.

[1] Available as open-source at : github.com/ETH-IVIA-Lab/VACP

3 Knowledge and Application State Access in VA

Building a gateway that lets AI agents perceive and operate within interactive VA applications requires identifying their functional components. Drawing on HCI and Information Visualization research [5,8,17,27,71], we decompose VA interfaces into two operational dimensions: *Knowledge Representation* and *Interactive Functionality*. This decomposition defines the state space an AI agent must perceive to reason effectively about the analytics process, and we analyze it to identify challenges (*chXX*) for providing AI agents accessibility in VA interfaces.

Knowledge Representation and Visual Composition — The primary function of VA interfaces is to represent knowledge, serving as a visual proxy for the underlying data model. At its core, a VA interface consists of at least one visualization view whose graphical elements encode data attributes as visual tokens (e.g., marks). Modern VA tools often compose multiple views into complex interfaces [5]. For an AI agent, understanding such an interface takes more than detecting individual visualizations: it requires parsing the structural logic or "grammar" of how views are spatially arranged and logically linked to represent high-dimensional knowledge (*ch01*: **Understanding Interface Logic**).

Interactivity, Operational Dynamics, and Feedback — Beyond static representation, the interface is defined by its Interactivity and Interaction Feedback. Literature in VA describes interactions abstractly in terms of user intent rather than technical implementation in web interfaces [27,37,41]. Following Munzner's What-Why-How framework [41], these interactive elements (lasso, zoom handles, or sliders) operationalize the user's or agent's intent (the "Why"). As Heer and Shneiderman [27] note, they drive transitions between data states (e.g., filtering) and visual states (e.g., encoding changes). When using VA interfaces, AI agents must identify and interact with these actuators to conduct or steer analysis (*ch02*: **Affordance Identification**).

High-level analytic interactions consist of sequential low-level input events (e.g., mouse events), and executing operations such as brushing a specific data cluster demands highly accurate spatial coordination. AI agents must therefore not only decide what action to take but also synthesize and execute these fine-grained, coordinate-based steps to reliably update the data state (*ch03*: **Accurate Interaction**).

Effective interaction requires perceptible responsiveness [47]. To close the interaction loop, VA systems provide visual feedback (e.g., highlighted selections, tooltips) confirming how actions change the system state [37]. This is critical for an AI agent: to interact reliably, it must verify that its actions have been successfully registered and processed by the application (*ch04*: **Interaction Feedback**).

Knowledge Layers —Data-rich, interactive VA environments pose distinct challenges for autonomous AI agents beyond general web navigation. VA application states are transient and multifaceted, and many interfaces rely on users' intuition and prior knowledge to discover interactions. While some interfaces provide tutorials [5], many assume implicit expectations and user expertise. Because LLM-based agents only know what appears in their training corpus, they need to be informed with the necessary knowledge and possible implicit expectations about VA interactions (*ch05*: **Providing Context**).

The "correct" interpretation of knowledge and interaction possibilities depends strongly on the interplay among the underlying data distribution, the user's analytical task, and the provided information.

Prior work [39,66] shows performance to scale with the structural clarity of the environment. However, there is no standardized context protocol for an agent to query the "holistic state" of a VA system across knowledge representation layers to interact with the interface. Moving from simple UI automation to collaborative analytics thus requires a unified representation that synthesizes perceptual, structural, and semantic knowledge (*ch01*). We identify four interlinked VA knowledge representation layers (see Fig. 2): L1 pixel-based, L2 DOM-based, L3 declarative grammar-based, and L4 production logic-based.

3.1 L1 Pixel-based Representation

Definition and Scope. In VA, the most practical knowledge representation is the rendered (usually 2D) interface that can be visually perceived. In web-based VA applications, this is the rendered DOM,

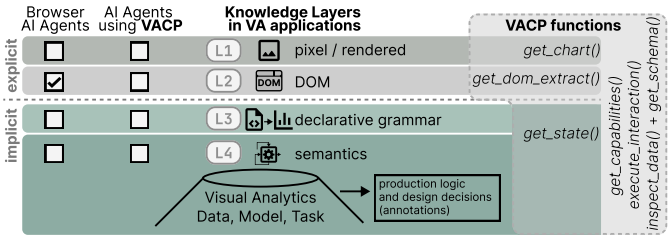

Fig. 2: Overview of knowledge representation layers in VA applications, agents' accessibility, and VACP functions for perceiving the VA interface at each level and supporting interactions. Each layer abstracts the knowledge, including the data to be analyzed, data encoding, and interface functionalities defined by production logic and design decisions.

i.e., the final product. Within the VA design process, it is the highest level of knowledge abstraction, summarizing all information visually. When data or visualization logic changes, this representation must be re-rendered. ***Format and Structure.*** The pixel-based representation is unstructured. Unlike the underlying layers (L2 – L4, see Fig. 2), it can only convey as much information as there are pixels on the screen. ***Encoded Knowledge and Added Value.*** Despite being unstructured, this representation can extract the application layout and the position of visual elements. In data visualizations [5], it reveals visually detectable data patterns, such as clusters, outliers, trends, high-density regions, and differences among data points via distinct visual encodings (e.g., color, shape). ***Interaction Capabilities.*** Many VA systems implement interactions for further exploration, but in the pixel-based representation, these are usually not explicitly described. Unless the visual interface explicitly lists and describes them (e.g., in text or explicit affordances), users and agents must infer available interactions from experience and implicit expectations (*ch02*). ***Agent Accessibility.*** An AI agent can access this layer by taking a screenshot of the current application interface. To interpret it, the agent needs computer vision capabilities (*ch06*: **Modality Access**).

3.2 L2 DOM-based Representation

Definition and Scope. In web applications, the Document Object Model (DOM) is a browser-managed logical tree representing the application's structure, element roles, and content. It abstracts VA knowledge from visual grammar and the developer's production logic, capturing the interface's live state for rendering and functionality. ***Format and Structure.*** The DOM is typically expressed in the text-based, XML-structured format—HTML—that defines the application's component hierarchy. Tags can contain text and code. The DOM derived from HTML is dynamic and can be modified with JavaScript. ***Encoded Knowledge and Added Value.*** The DOM layer adds value beyond the pixel-based representation. It maintains the L1 knowledge representation but augments it with retrievable data values, exact numbers, and precise positions of visual marks. It supports nested hierarchies and configuration components, such as form elements mapping data attributes to axes. Through metadata and accessibility tags (e.g., aria-attributes [29]), it also reveals how visual variables are encoded and provides additional descriptions of data attributes that a pure image cannot convey. ***Interaction Capabilities.*** The DOM includes event listeners (e.g., for mouse events) that define interactivity. From these listeners, agents can infer baseline interactions such as `click` and `hover` (*ch02*). Higher-level interactions, such as brushing—a sequence of mouse-down, mouse-move, and mouse-up—may also be inferred when tags have specific attributes or names, though this is not guaranteed. ***Agent Accessibility.*** AI agents must access the DOM at runtime (*ch07*: **Sync States**) and process its textual representation.

3.3 L3 Declarative Grammar Representation

Definition and Scope. A declarative grammar describes the interface's format through rules or guidelines for creating consistent, effective visualizations. Such grammars define a common vocabulary and structure for building and interpreting visualizations. In the VA application design process, this is a low-level abstraction: everything is strictly specified by the template schema before compilation into layer L2.

Format and Structure. Common grammars such as Vega, Vega-Lite, and Mosaic's vgplot use `JSON` or `YAML`. These definitions serve as blueprints for the interactive visualizations in the rendered application. ***Encoded Knowledge and Added Value.*** Based on these definitions, this layer encodes knowledge of attribute visualization, hierarchies, layouts, and rules for color selection and chart types. It explicitly maps data attributes to visual objects. Unlike the DOM, which represents the computed visual components, the declarative grammar captures the precise intent and configuration of the visualization design. ***Interaction Capabilities.*** Interactions are explicitly defined and linked to specific data attributes and visual objects in the grammar. Higher-level interactions like brushing are directly declared, listing available interactions. Additional semantic knowledge can be inferred from the interaction definitions and the data variables they manipulate. (*ch05*) ***Agent Accessibility.*** The declarative grammar-based definitions must be injected into the runtime to be accessible to an AI agent (*ch07*) via specific tools or DOM access. As with layer L2, an AI agent needs text-parsing capabilities to interpret (semi-)structured text schemas.

3.4 L4 Semantic Knowledge Representation

Definition and Scope. The production logic encodes semantic knowledge generated during VA design and development. While layers L1 – L3 describe composition and outcome, layer L4 captures the "why": design decisions, rationales, and intended purposes of specific interactions or visual marks, offering a holistic application view. ***Format and Structure.*** As these details can be any kind of information, there is no fixed format or template. When available, they are usually documented as free-text annotations by the developer, such as code comments. ***Encoded Knowledge and Added Value.*** This semantic knowledge reveals insights into the underlying data, machine learning model, and intended tasks or user groups. It explains why a data attribute is encoded in a specific way and whether a VA application suits particular tasks. ***Interaction Capabilities.*** This layer's interaction knowledge is semantic rather than functional. It describes metadata, such as an interaction's purpose, instead of just technical event listeners or grammar definitions. ***Agent Accessibility.*** VA applications do not naturally expose this knowledge layer to users or agents, requiring extra logic or methods to access it at runtime. Because the knowledge is mostly encoded in free text, AI agents require robust text parsing and natural language understanding to interpret the semantic information (*ch06*).

4 VACP: The Visual Analytics Context Protocol

To bridge the gap between human-oriented VA interfaces and AI agents, there is need for a protocol that meets both their requirements and mediates between them. The VA application requires strict system logic and state consistency, while AI agents need semantic richness and a way to perceive and act on the VA interface. We, therefore, propose the **Visual Analytics Context Protocol (`VACP`)**.

VACP serves as a formal abstraction layer that decouples the agent from implementation details (e.g., DOM structures or pixel data), while preserving a holistic semantic understanding of the analytical environment. It comprises three components: (1) **Application State Representation**, modeling the interface and its elements as a temporal semantic graph; (2) **Exposition of Available Interactions**, formalizing valid state transitions; and (3) **Interaction Execution Gateway**, translating the agent's intent into executed interactions. We ground our context protocol in principles that we consider the minimal requirements for enabling agents to navigate data-rich environments, even beyond VA applications. Some principles inform the agent through Contextualization, while others ensure operations and Interactive Dynamics in the VA interface. In the following, we first introduce the principle, then explain its application in VACP and annotate the challenges (identified in Sec. 3) that we address with the principle.

4.1 Application State Representation

P1 – App State Exposure Application State Contextualization
To reason about an interface, the agent needs an up-to-date, semantically grounded representation of the application state.

Unlike experienced human VA users, who intuitively grasp visual metaphors, AI agents require explicit encoding of component purposes and the developer's design decisions. We encode the typical hierarchical structure of VA application in the state graph. **Nodes** represent distinct VA components from global settings and dataset selection to fine-grained elements like interaction feedback. **Edges** capture structural containment (e.g., a "Dashboard" node containing a "Scatterplot" node) and semantic relations, such as coordinated views (e.g., "Selection in Plot A filters Data in Plot B"). In VACP, agents access the (semantic) application state via *get_state()*, and layers L1 and L2 via *get_chart()* and *get_dom_extract()*. (*ch01*, *ch07*)

P2 – Metadata Annotation Metadata Contextualization
The reasoning barrier for agents can be lowered by adding descriptive metadata about component purpose or behavior.

To support reliable reference and reasoning, the state representation adheres to the following constraints: **Unique Identification:** Each node has a stable, unique ID. So, the agent can consistently reference components across a session. **Temporal Consistency:** The application is treated as a living document. Each state snapshot includes a timestamp and unique snapshot ID to avoid race conditions and ensure the agent uses the latest interface state. **Semantic Annotation:** Nodes are annotated with metadata about component type, semantic layer, and a natural language description of their purpose. Optional metadata further enriches context and eases the agent's reasoning. (*ch05*)

P3 – Agent Constraint Awareness Interactive Dynamics
Considering agent constraints, such as context window size, optimizes agent runtime and reduces the risk of performance degradation.

Crucially, VACP addresses the finite context window of LLM-based AI agents with a details-on-demand architecture. Instead of serializing all data into state and hitting token limits, the state only exposes the data's *structure* and *availability*. The agent can use *inspect_data()* to retrieve only the specific data points relevant to its current task. (*ch05*, *ch06*)

P4 – Provenance Tracking Temporal Contextualization
Providing direct jumps back to a previous state improves the performance of the agent. Provenance tracking is optional.

When interacting with the VA interfaces, agents might reach a dead end. Reverting to previous states might require multiple interactions. This can be overcome by allowing jumping to previous states.

4.2 Exposition of Available Interactions

Interactive VA systems are defined by the dynamic relationship between the user and the system. In VACP, an **interaction** is an intent-driven agent action that triggers a system reaction, resulting in a mutation of the application state. We formalize the state updates as follows. Let $V = \{v_1, \ldots, v_n\}$ be the set of visual components and $I = \{i_1, \ldots, i_m\}$ the set of available interactions. The application state at time T, S_T, is defined by the configuration of these components and available interactions: $S_T = \{V_T, I_T\}$. Applying an interaction $i_{x_T} \in I_T$ with parameters $P = \{p_1, \ldots, p_k\}$ produces the next state: $S_{T+1} = i_{x_T}(S_T, P)$.

P5 – Interaction Exposure Interaction Contextualization
All currently valid interactions need to be explicitly exposed to the agent. This exposure is dynamic.

To enable the agent to observe the available interaction capabilities, VACP requires registering available interactions at runtime. For safe execution, the protocol specifies interaction parameters, indicating which are required or optional, their data types, and valid ranges. Additional descriptions provide functional context. Agents can query all available interactions via *get_capabilities()*. (*ch02*)

P6 – Dynamic Interaction Update Update Dynamics
Upon an application state update, the protocol must dynamically update the list of capabilities to reflect the currently available interactions.

To address the dynamics of the systems, VACP allows for dynamic registration or release of interactions to reflect the current context. Interactions are represented as **action nodes** within the state graph, linked to the components they manipulate. (*ch02*)

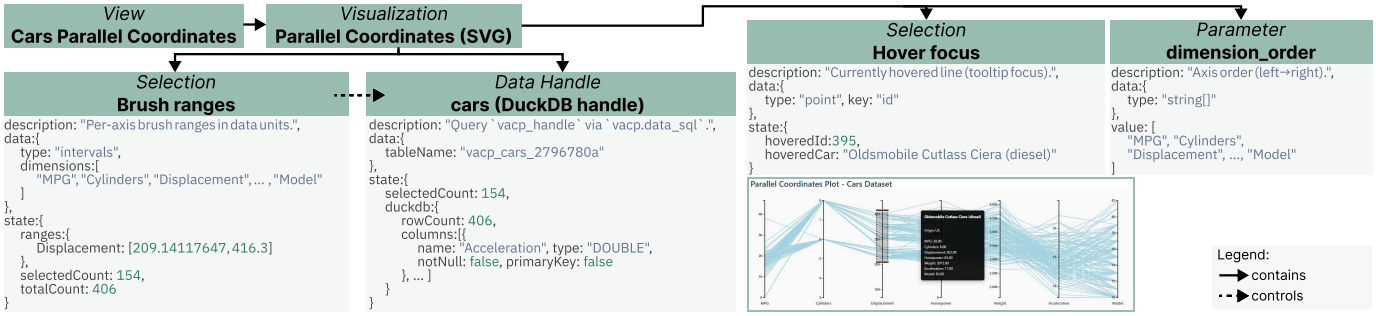

Fig. 3: Example VACP application-state representation. The capability graph captures semantic structure and interaction-relevant relations; the state snapshot stores currently active values under the same stable references.

P7 – Tailored Data Access Data Contextualization

Instead of serializing all underlying data into state, which would rapidly exhaust token limits, the state exposes only the *structure* and *availability* of data. The agent then queries specific data points only when they are relevant to the current task.

Additionally, VACP mitigates data-access bottlenecks with two actions: *get_schema()*: Lets the agent inspect the data's logical structure and constraints without loading records. *inspect_data()*: Lets the agent run structured queries to retrieve subsets or aggregates of data tied to a visual component, delegating arithmetic to the system engine and avoiding LLM arithmetic calculation errors. (*ch06*)

4.3 Gateway for Direct Interaction Execution

P8 – Interaction Execution Gateway Execution Dynamics

To close the perception–action loop, the agent requires an API to execute interactions, giving it full accessibility beyond view-only mode and allowing it to manipulate the application state.

The final VACP component is the execution gateway, which links perception to action and serves as the operational API for the agent's action calls. When *execute_interaction()* is invoked, the gateway runs multi-step validation: it resolves the interaction reference ID to the VA system function, verifies that the interaction is still valid for the current state snapshot (handling latency-related conflicts), and checks that the provided parameters satisfy the type and range constraints defined during exposition. The gateway executes the interaction synchronously, immediately applying state updates to minimize execution time. The agent receives structured feedback. On success: a confirmation status or, for data queries, the requested results. On failure: a descriptive error message that aids agentic reasoning, helping diagnose issues and attempt self-correction. The gateway can also support advanced mixed-initiative features such as interaction provenance logging, undo/redo stacks for safe exploration, and "human-in-the-loop" hooks that require user confirmation for high-stakes actions. (*ch02*, *ch03*, *ch04*)

4.4 Agent Workflow using VACP

Fig. 1 shows how AI agents can use VACP to interact with a VA interface. The agent first inspects the application state by interpreting the interface structure, VACP metadata, and interactive elements. It then chooses an action to advance its analysis, such as changing a visual data mapping, which updates the interface. The agent reads the updated state via VACP to verify the change. To focus on a data subset, it can brush the line chart. The system converts this into a selection, updates the application state, reflects the selection in the interface, and VACP passes this updated state back to the agent for its next step.

5 The VACP Library

We implement a TypeScript library of VACP for web-based VA systems[1]. The library decouples the semantic interaction contract from the DOM layout, rendering backend, and UI framework. To integrate VACP with existing ecosystems, we provide adapters for declarative grammars such as Vega-Lite [52] and Mosaic's vgplot [25], an API for custom solutions, plus an MCP server for transport [4].

Design Goals — The implementation targets four goals: *Minimal setup* via default capabilities and state mappings; *Direct integration* with common VA stacks to retrofit existing interfaces without changing interaction logic; *Extensibility* through domain-specific metadata and custom actions; and *State consistency* by synchronizing a single application state across human and agent interactions.

5.1 Semantic Grammar: State and Interactions

The architecture isolates protocol semantics from transport layers. At runtime, the application defines three synchronized objects: a capability graph, a state snapshot, and an action catalog. Transport layers pass requests to this runtime without changing the semantic meaning.

State Representation — The library divides application state into structural capabilities and volatile values (Fig. 3). Structural capabilities are mostly static and define the semantic space via the capability graph. Volatile values change during interaction and encode current selections or control settings in the state snapshot. Both layers use stable semantic references, letting agents track entities across multi-step interactions. Developers can optionally attach metadata to clarify domain semantics when multiple valid operations exist.

Action and Data Representation — VACP models interactions as semantic actions instead of low-level UI events. Each catalog action has a stable identifier, parameter schema, and optional target reference. VACP offers structured queries on referenced data handles, letting agents retrieve relevant datapoints and aggregates without reading from pixels or the DOM. This bounds data extraction and avoids serializing full datasets into the LLM context.

5.2 Execution and Transport Loop

Interaction execution uses a validated request–response cycle. When an agent submits an action, the runtime validates parameters, runs the framework logic, and returns structured feedback. This explicit channel surfaces execution errors for self-correction. Afterward, the agent queries the updated state via stable references to verify the outcome.

To reduce LLM context limits and execution time, the implementation avoids continuous full-state serialization, scoping state updates to semantic references and returning them as incremental diffs.

MCP Agent Accessibility — We provide an MCP server that acts exclusively as a transport layer for VACP, exposing a small, static set of core tools for state observation and action execution. Since updating the MCP tool registry after each UI change would be too costly in tokens and execution time, we deliberately avoid dynamically mapping VA interactions to MCP tools. The agent uses the tools to query the internal VACP action catalog and decides when to refresh its view of available

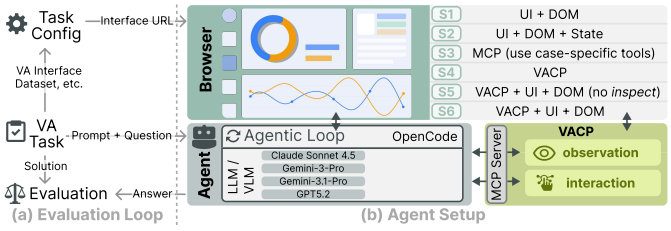

Fig. 4: Overview of the Evaluation pipeline, where the left (a) shows the overall task execution and evaluation workflow, and the right (b) illustrates the agent setup and the different evaluated scenarios S1 – S6.

interactions. If it attempts an obsolete action, the VACP runtime returns a structured error, prompting the agent to adjust.

5.3 Integration Examples

The library uses adapter patterns to map framework-specific interactions to the shared VACP contract, supporting two common VA paradigms: declarative grammars and custom imperative visualizations. In Tab. 1 we show the implementation effort for our use cases.

Declarative libraries: an adapter parses the specification to extract encodings, parameters, and selections. A wrapper maps native Vega-Lite specifications to protocol actions while preserving execution logic.

```
import { installVacpOnVegaLiteView } from '@vacp/vega-lite';
import vegaEmbed from 'vega-embed';

// 1. Render the standard Vega-Lite chart
const containerId = '#chart-container';
const { view } = await vegaEmbed(containerId, vegaLiteSpec);

// 2. Wrap the chart to connect it to the AI agent
installVacpOnVegaLiteView({
   root: document.getEtementById(containerId),
   view: view,
   spec: vegaLiteSpec,
});
```

Custom charts: without a standardized internal model, developers manually build the semantic bridge by defining stable references to interactive components, exposing framework-specific functions as semantic actions, and adding a polling function to sync live state.

```
import { createVacpActions, createVacpRefBuilder, installVacp } from '@vacp/kit';

// 1. Create IDs for the chart and its brush tool
const refs = createVacpRefBuilder({ appId:'demo', viewId:'custom' });
const [ vizRef, brushRef ] = [refs.root, refs.ref('/selection/brushes')];

// 2. Define what the AI agent can do
const actions = createVacpActions()
   .action('set_brush','Set axis brush ranges')
   .target(brushRef).params({ /* Omit parameter details */ })
   .handle(p => {chart.setBrushRanges(p.ranges);return { ok: true}});

// 3. Connect the chart's structure and live state to the agent
installVacp({
   actions: actions.registry,
   graph: {
      nodes: [{ ref: vizRef, kind: 'Visualization', title: 'Chart' },
              { ref: brushRef, kind: 'Selection', title: 'Brush' }],
      edges: [{ from: vizRef, to: brushRef, kind: 'contains' }]
   },
   getState: () => ({
      // Provide the current selection back to the agent
      [brushRef]: { kind: 'Selection', ranges: chart.getBrushRanges() }
   })
});
```

6 Evaluation

We evaluate VACP-enabled agents against state-of-the-art approaches to validate the VACP framework's effectiveness and improvements. We detail our methodology, evaluation setup, and findings.

6.1 Methodology

To systematically evaluate and improve AI web agents in VA, we established a multi-stage process encompassing the curation of a proof-of-concept benchmark dataset, expert task validation, identification of agent limitations, and the development and evaluation of VACP.

Benchmark Development and Task Formalization — While visualization literacy tests like VLAT [35] exist, to the best of our knowledge, there is no standardized benchmark for assessing users' ability to use interactive VA systems to solve problems. Addressing this, we curated a task set of baseline VA systems and tasks from established literature, such as visualization design spaces and literacy tests, that experts validated as individually indicative of prominent domain practices. We prioritized atomic tasks rather than complex workflows to isolate specific perception and execution failures.

To reflect on the diversity of web technologies in real-world VA systems, we implemented the test systems using two distinct approaches: custom, low-level code (React and D3.js) and high-level specification languages (Vega, Vega-Lite, and Mosaic vgplot). This variety allows testing whether an agent can adapt to different web structures and levels of semantic complexity. The final tasks suite (see Sec. 6.2) was refined through iterative expert discussions among the authors.

Table 1: Required lines of code (LoC) for adding VACP to use cases.

use cases	framework	total (LoC)	semantic (LoC)	action (LoC)
UC1, UC2, UC4	Vega-Lite	37	0	0
UC3	Mosaic vgplot	42	0	0
UC5	React + D3	395	47	224

Pre-Study and Expert Validation — To ensure our benchmark was solvable and represents real VA scenarios, we conducted a formative pre-study with VA experts, who interacted with the selected VA systems to solve the defined tasks. The study aimed to (1) verify task solvability, (2) assess the suitability of the visualizations and interactions, and (3) evaluate the representativeness of system components. It yielded critical feedback for optimizing our experimental design. Experts confirmed that the tasks, visualizations, and interactions represent real VA challenges but identified ambiguities in several task descriptions. We refined these descriptions to ensure that subsequent agent failures could be attributed to functional limitations rather than prompt ambiguity.

Baseline Agent Evaluation — We evaluated current state-of-the-art web AI agents on our benchmark and found that they fail most tasks (completion rate 28%–73%). Agents miss the visualization's semantic context, struggle to access data details hidden within the DOM, and lack the precision for accurate interactions.

Comparative Evaluation — We evaluated the efficacy of our approach by implementing VACP and giving AI agents access to different context scenarios. This summative evaluation tested whether agents using VACP could complete the analytic tasks. We compared VACP against alternative context types, measuring task completion, duration, and token usage. For deeper analysis, we annotated each agent turn with dimensions based on foundational HCI literature [10,42,48] to quantify agent behavior, recognizing that solution paths are highly iterative and that successes and failures are not random prompt artifacts, but rather structural consequences of navigating or failing to navigate the VA pipeline based on the provided knowledge and interaction access.

6.2 Use Case and Task Description

To rigorously evaluate AI agents in VA, we used the previous curated set of compelling use cases spanning a comprehensive design space. The scenarios cover diverse data modalities, visualization techniques, and complex interactions from existing taxonomies [5,8,41]. The use cases span all core data types from Munzner [41] instantiated within a targeted visual representation space of numbers, tables, and detailed visualizations [5]. Available interactions include select, navigate, arrange, change, and filter, excluding aggregation, introduction, and dashboard-reconfiguring encoding tasks (see [8], How). To probe technical limits, we include both declarative grammars and imperative paradigms (React, D3.js), requiring agents to manipulate SVG-based DOM elements and pixel-based canvas renderings (see Fig. 5).

UC1 – Global Development: This interactive bubble chart visualizes health demographics over time. It supports temporal animation via sliders, tooltips for details, and geometric zooming/panning to resolve occlusion in dense regions. **UC2 – Seattle Weather:** A coordinated multiple-view system links a temperature scatterplot with a weather-frequency bar chart. The tightly coupled views support bidirectional cross-filtering—temporal brushing updates distributions, categorical selection filters records, and hover reveals precise details. **UC3 – Olympic Athletes:** A dashboard combining a biometric scatterplot (with gender-based regression lines) and a sortable data table, supporting complex queries via spatial brushing and filters for sport, gender, and keywords. **UC4 – U.S. Flights:** A geospatial node-link diagram

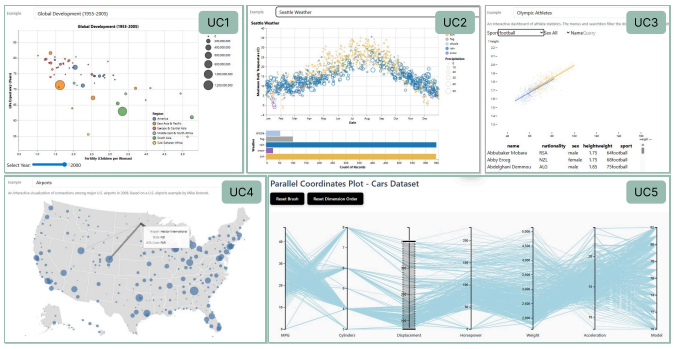

Fig. 5: Overview of VA interfaces of the defined VA use cases.

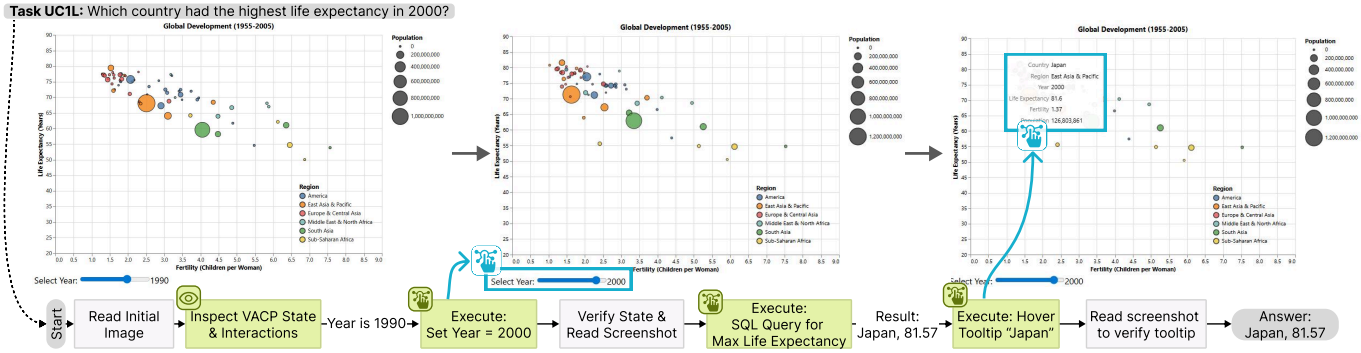

Fig. 6: Claude Sonnet 4.5 solving Task UC1L in S6. After inspecting a screenshot and the VACP semantic state, the agent adjusts the selected year and verifies the change in the UI. It then queries the data, identifies and confirms "Japan," by hovering the data point, and returns the correct answer.

overlaid on a map supports egocentric network analysis. Clicking an airport node filters outgoing flight edges, while hovering reveals weighted edge attributes (flight counts) and node metadata. **UC5 – Technical Car Specs:** A Parallel Coordinates Plot for high-dimensional automotive data [49]. It enables multivariate analysis through multi-axis brushing for concurrent constraints, drag-and-drop axis reordering to expose correlations, and global reset controls.

Task Definitions — We define the tasks using Brehmer and Munzner's multi-level typology [8]. From the Why dimension, our evaluation targets locate, identify, and compare actions, excluding produce and summarize. Agents must use latent lookup, browse, and explore behaviors as primitives to obtain correct answers. For each use case UCx, we specify three tasks (UCxL, UCxI, UCxC) that differ in required interactions and focus on elementary analytical tasks: *locate* (UCxL), *identify* (UCxI), and *compare* (UCxC). Some tasks involve multiple elementary actions, but the primary one is indicated by the index. Ground truth solutions are derived from the underlying datasets and results are validated via semantic similarity (textual values) and numeric tolerances (numerical values) defined in the evaluation setup. All task descriptions, mappings to taxonomies, solutions, and the validation setup are provided in the supplemental material.

6.3 Validation of Tasks & Interfaces

To align our benchmark with real VA challenges, we conducted an expert user study to verify that our use cases and tasks form a representative baseline for evaluating AI agents.

Study Design and Demographics — We recruited 7 participants (Pt) (six domain experts in VA and HCI, and one non-expert) to complete 15 tasks across our VA use cases and to assess expert alignment and interface intuitiveness. Participants rated cognitive demand and tediousness, and judged task representativeness on a 5-point Likert scale.

Representativeness of Interactions and Tasks — Participants rated the interactions as highly representative (Median $= 5, \mu \approx 4.6$). Pt4 described them as "very natural interactions," mirroring how a human analyst explores data. Pt1, Pt2, Pt5, Pt6, and Pt7 agreed that "all interactions made sense." Participants rated the tasks as highly representative (Median $= 4, \mu \approx 4.4$). Pt4 noted the tasks cover a wide range, including diverse datasets, variations, temporal data, and geospatial patterns. Pt1 said real VA tasks are "more complex and open-ended" than the atomic tasks, but we emphasize that these are the basic building blocks of analysis. AI agents struggling with foundational tasks are not ready for the ambiguity of real-world exploration.

Task Correctness and Ambiguity Resolution — Human participants achieved an 85.71% completion rate. No task stumped all participants, confirming solvability. The study revealed specific challenges that informed our final task descriptions: *Visual Complexity:* UC4 is difficult because dense datapoints create overlapping lines, obscuring routes. This shows the testbed captures real-world "messiness," where occlusion challenges agents. *Ambiguity:* Pt2, Pt4, Pt5, and Pt6 flagged ambiguous terms in UC1 (finding the "largest" country) and UC4 (defining "smallest" airport) use cases. We revised prompts to remove ambiguity while keeping visual difficulty. *Precision vs. Estimation:* Accuracy varied by strategy: participants using hovers and tooltips were more precise than those visually estimating from axes. This distinction is crucial for evaluation, separating agents that can "read" the DOM from those that must "see" the rasterized chart.

Interface Design and Usability — Including a non-expert (Pt5) helped test usability. Despite lacking VA expertise, Pt5 identified interactions and completed tasks, indicating intuitive interfaces. Pt3 felt that some visual encodings could be improved, but said that such imperfections are common in real-world interfaces. Pt5 requested "more precise controls, like entering explicit numbers or text for filtering," supporting our VACP contribution, which gives agents more precise control options than standard interfaces.

Summary — This study shows our testbed is realistic for VA agent testing, with natural interactions and tasks that accurately reflect real visual and cognitive challenges in VA workflows.

6.4 Agent Evaluation

To validate the VACP framework, we evaluated AI agent performance under varying levels of contextual access, examining how different interface representations (from standard visual scene captures to our semantically enriched VACP interface) affect VA task completion.

6.4.1 Evaluation Setup

We utilize multiple AI agents composed of multimodal foundation models for text parsing, vision, and tool use, operating within a standardized agentic loop. This loop follows the ReAct [68] framework, allowing agents to iteratively reason, act, and observe feedback. For evaluation, we used human pre-validated VA tasks and interfaces (see Sec. 6.3).

Models and Sampling Strategy — To avoid artifacts tied to a specific model, we evaluate four SOTA models: *GPT-5.2* (OpenAI) [45], *Claude 4.5 Sonnet* (Anthropic) [3], *Gemini 3 Pro Preview* and *Gemini 3.1 Pro* (Google) [20]. Due to the non-deterministic nature of LLMs (e.g., temperature), we ran all experiments three times per model to capture variance in success rate, token usage, and runtime. This allows us to assess generalizability and identify cross-model behavioral patterns across scenarios. While runs differed in their reasoning paths, they produced similar outcomes, with only minor variation in token consumption and duration. We added *Gemini 3.1 Pro* to our evaluation as *Gemini 3 Pro* was deprecated when running ablations S3 and S5.

Scenarios — We evaluate agents in six scenarios, reflecting increasing context, knowledge representation, and interaction access:
S1 **UI + DOM:** Mirroring current general-purpose web agents, agents see only screenshots (L1) and raw DOM (L2). Complex visualizations (Vega-Lite/Mosaic) are rendered as canvas elements, so their data is hidden from DOM parsers. Only the PCP use case uses SVG. Agents act via emulated mouse events and JavaScript execution.
S2 **UI + DOM + State:** Building on L1 and L2, we inject the instantaneous VACP application state into the DOM as serialized JSON in a hidden node. Agents can discover and access it via DOM queries.
S3 **MCP (with *inspect*):** To compare VACP from simpler structured tool access, in this ablation, agents interact *exclusively* via use case-specific static tools from an MCP server. While providing the state, interactions, and *inspect_data()*, there is no visual and DOM access.
S4 **VACP (State + Interaction Gateway):** This ablation removes visual and DOM access. Agents interact *exclusively* via tools from the VACP MCP server (L3, L4), relying entirely on the semantic API and

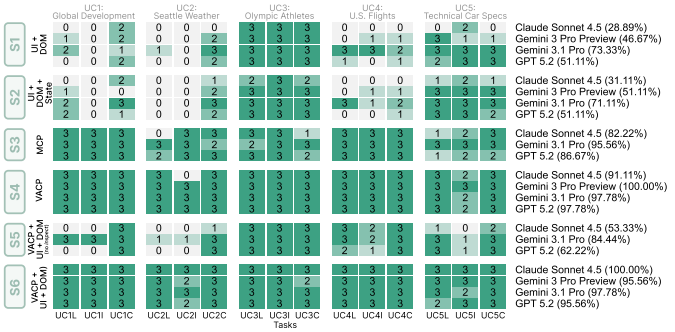

Fig. 7: Summary of the AI agent evaluation on the provided use cases and tasks. Each tile presents the number of successful runs over three runs. Percentages after the agent model name indicate overall success rate on the provided scenarios S1 – S6, use cases, and tasks.

testing the VACP schema's expressiveness without visual grounding.

S5 **VACP + UI + DOM (no *inspect*):** Agents receive the full toolset (L1 – L4), except *inspect_data*): This lets us study the contribution of interaction grounding and data access through *incpect_data()*.

S6 **VACP + UI + DOM:** Agents receive the full toolset (L1 – L4): VACP tools, DOM access, and screenshots, letting us study synergies or conflicts between visual perception and semantic data retrieval.

Technical Implementation — We use OpenCode [1] to manage the agentic loop, ensuring a robust, reproducible tool-execution environment. Each experiment runs in an isolated, sandboxed browser instance. Fig. 4 shows the evaluation setup. All agent runs are limited to a max duration of 20 minutes without allowing for retries and share the same system prompt, which specifies the task, expected response format (see supplemental material), and the initial interface URL. The system prompt was not updated during the runs. We initialize all models with temperature 1.0 (recommended by the model providers). In scenarios S4, S5 and S6, agents access VACP via an MCP server exposing the available tools. All experiments were run on an Apple MacBook Pro with an M3 Max chip and 36GB memory, using model providers' APIs for inference. The evaluation source code and details on API and models are provided in the supplemental material.

Initial Test Runs — Prior to the main evaluation, we conducted pilot studies with a standard "computer use" loop [22]. Performance was suboptimal: agents struggled with controls such as dropdowns and inputs, and the setup did not support non-Gemini models. This resulted in the shift to the more robust, well-tested, open-source OpenCode framework. We also found that agents exploited "loopholes" instead of solving VA tasks, e.g., hallucinating answers from pre-training on standard datasets, downloading data from external GitHub repositories, or writing Python scripts to parse the DOM and bypass visual interaction. In response, we refined the system prompt to enforce strict tool use and restricted sandbox network access to prevent data leakage.

6.4.2 Evaluation Results

Fig. 7 shows our experiment summaries. Detailed traces of agents' perception, chain-of-thought, and actions are in supplemental material.

Success Rates and the Visual Gap — We observe a performance gap between visual-based and semantic-based agents. Scenario S1 yielded the lowest success rates (28%–73%) across all models: agents struggled to extract precise values from pixel-based canvas visualizations and DOM representations, confirming that current multimodal models are inadequate for precision-critical VA tasks. Adding the state to the DOM (S2) slightly improved success (31%–71%, max. gain 4.4%). In contrast, scenarios relying on VACP's semantic interface (S4, S6) achieved near-perfect success across tasks. Analysis of error types (**??**) shows that in S1 and S2 the main failures stem from agents not fully exploring relevant data or giving up after several interaction attempts. Agents with structured tool access (S3) fail by misinspecting data, encountering execution timeouts, and not completing required interactions. In S4 and S6, failed runs mostly result from misinterpreting or incorrectly requesting data. While succeeding in S4, one agent returned an unparsable response in S6, and another failed while deciding to reverse-engineer data from the DOM. Without *inspect_data* (S5), most failures are visual errors similar to those in S1 and S2. Thus, providing a semantic interface significantly improves success rates, indicating that structured semantic state representations and API-driven interaction are more reliable for analytical reasoning and interaction than purely visual or textual DOM interpretation.

Efficiency and Token Consumption — Interaction modality also affects computational efficiency. Agents in scenario S4 (VACP) consumed vastly fewer tokens and finished faster than those in S1, S2, S5, and S6 (Fig. 8). Agents provided with use case-specific tools (S3) consume the fewest tokens but have lower success rates. Removing DOM parsing and high-dimensional image processing allowed for leaner reasoning traces and interactions. The hybrid scenario S6 enabled "visual verification" (double-checking actions against screenshots), which consistently increased token usage and time on task, without a proportional increase in success rate. *Claude Sonnet 4.5* agents consumed far fewer tokens on average (<2000 tokens for S4) than ones with other models (e.g., >30000 tokens for S4 on GPT-5.2).

Added Value of Structured Tool Access and Data Inspection — While consuming slightly more tokens (Fig. 8), VACP provides value over simpler use case tailored structured tool access by achieving higher task completion rates (S4: 97%, S3: 89%). Further, in S3 for two of the five "locate" tasks, where the visualizations are cluttered, the agents fail to visually perceive the values, as they estimate the values or use wrong data points as reference. In ablation S5 we study the impact of direct data access. When restricting the data inspection (S5), the interaction grounding in the additional semantic metadata leads to higher completion rates than for the baseline (S1) and state injection (S2), but has lower success rates on "identify" tasks than on the other tasks, as they fail to read the exact values. Comparing S5 and S6, agents with tools to inspect the data have improved completion rates while reducing the token consumption and execution time.

Agent Behavior and Interaction Stata/Analytic Provenance — While efficiency metrics highlight the computational benefits of semantic interfaces, explaining these performance gaps requires examining agents' iterative problem-solving strategies. To confirm that success rates stem from the provided context rather than prompt artifacts, we annotated every agent turn across four dimensions (*Stratum*, *Phase*, *Outcome*, *Adaptation*) grounded in foundational HCI literature [10, 42, 48].

We operationalize these human-centric frameworks for LLM agents by mapping Norman's Gulf of Execution [42] to an agent's ability to formulate valid API calls or DOM interactions, and the Gulf of Evaluation to its capacity to parse returned state representations or pixels. We similarly adapt Pirolli and Card's sensemaking loop [48] to characterize how agents update their internal context windows from intermediate results. For each turn in an agent run, we annotate where the agent is operating (*Stratum*), the turn's intent (*Phase*), whether the *Outcome* was successful, and if the turn triggered a strategic *Adaptation*. Annotations were generated with a structured LLM approach. To ensure methodological rigor, the authors manually validated on a random 20% sample, achieving strong inter-rater reliability (Cohen's $\kappa = 0.918$, between the human annotator and the LLM). In case of disagreement, the annotation of the LLM was corrected. Because this work focuses on interface accessibility, we detail only the interaction stratum below. Full reports on intent, outcome, etc. appear in the supplemental material.

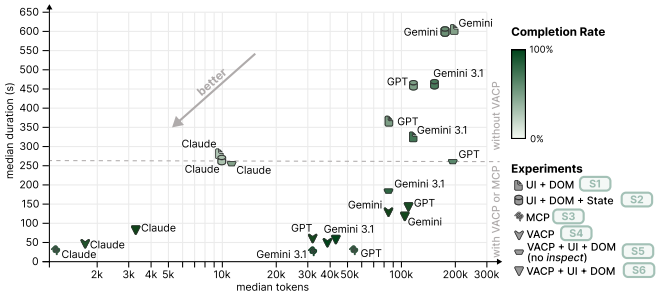

Fig. 8: Summary of the AI agent evaluation. Detailed analysis of median token consumption and task execution time for the different context scenarios, averaged on the use cases and evaluated models.

Interaction Strata and the Verification Loop — Following Buxton's taxonomy [10], we grouped interactions into three strata: pragmatic (manipulating raw pixels or coordinate-based events), syntactic (navigating the logical DOM or view structure), and semantic (interacting with raw data models, schemas, or analytical APIs). Their distribution shows a strong link between semantic access and task success. In S1, agents relied almost entirely on syntactic (avg. 48.44%) and pragmatic (avg. 47.19%) interactions, with minimal semantics (avg. 4.37%). Adding application state to the DOM in S2 slightly increased semantics (avg. 11.82%) and syntactic interactions peaked (avg. 53.12%), while semantic interactions saw a modest rise (avg. 11.82%). With only accessing the VACP interface in S4, agents operated almost entirely at the semantic level (99.8%). Scenario S6 presented the most revealing patterns. With full access to all layers, agents favored semantic interactions (avg. 64.86%) while still using a notable share of pragmatic interactions (avg. 29.04%). They typically used semantic interactions for initial insights, then dropped to the pragmatic layer to visually verify solutions before answering. Conversely, failed runs by GPT-5.2 in this scenario showed a collapse of this strategy with far fewer semantic interactions (25%) and over-reliance on the syntactic layer (55%).

Qualitative Strategies and Sensemaking Quirks — Beyond aggregate statistics, the qualitative traces reveal notable sensemaking behaviors. Fig. 6 shows a successful S6 run by Claude Sonnet 4.5, demonstrating a robust semantic-to-pragmatic loop: the agent selects the required year, queries the data semantically, and validates intermediate steps with screenshots. In visually constrained environments (S1), agents used creative hacks to bridge the Gulf of Execution. When Claude Sonnet 4.5 failed to perform reliable brush selection, it inferred the subset from SVG geometries and tried to visually confirm its selection by mutating DOM styles to recolor lines. Gemini 3 Pro Preview (S1) estimated weather patterns by falling back to pixel counting via color thresholds. With semantic access (S4), agents exhibited high-level analytical reasoning: Claude Sonnet noted the risk of visual misinterpretation and computed regression slopes directly from the data. But semantic access also introduced new pitfalls, such as GPT-5.2 failing by assuming it should only find IQR-based outliers and missing a second data point. In mixed environments, agents showed human-like corroboration, using SQL to identify the top country and then triggering a semantic hover tooltip to visually reconcile a rounding discrepancy.

Design Implications for Agentic VA — These behaviors signal a key shift in how we must design interfaces for AI operators. Agents' tendency to add pixel-approximation layers or alter the DOM for visual confirmation suggests that VA systems should expose intermediate analytical states instead of forcing agents to infer them from visuals. The effectiveness of the semantic-to-pragmatic verification loop indicates that purely "headless" VA is inadequate: agents need to confirm semantic queries with visual representations, as humans do when comparing tables with charts. Future agent-ready VA systems should therefore offer a dual-channel architecture: a robust, API-driven semantic layer for reliable execution, paired with deterministic visual feedback to support agents' internal evaluation and sensemaking.

Summary — Our results show that a tailored semantic interface like VACP is not merely an enhancement but a requirement for reliable autonomous VA, bridging the gap between pixel-oriented web agents and the data-centric needs of VA applications.

7 Discussion

VACP allows AI agents to interact with VA interfaces using the same mixed-initiative paradigm as human users [40] while maintaining human insights and control [15] by shifting from model-specific prompt engineering to structured context engineering. Our evaluations show that the structured exposure significantly increases task completion. As frontier models converge in reasoning and tool-calling, the critical bottleneck for agentic VA becomes how well systems expose semantic state and affordances. Process analyses reveal that agents can recover from local coding errors but fail predictably when lacking interface context. VACP addresses this by encoding system knowledge and interaction options into an agent-tailored format, providing a robust gateway for accurate analytical execution. VACP significantly reduces token usage relative to baseline approaches, lowering costs for future systems. Preliminary experiments with smaller, on-device LLMs using VACP also showed promising success, opening the door for local AI agents.

Limitations and Future Work — While establishing a robust foundation for agent-accessible VA, VACP's current design operates under several inherent conceptual and technical limits.

Proof-of-Concept Evaluation: VACP-enabled agents show high success rates, but our benchmark relies on elementary analytical tasks to isolate core perception and execution failures. Agents that struggle here are not ready for real-world, more complex VA scenarios. We therefore do not claim that VACP currently supports complex exploratory workflows; it instead provides the operational foundation. Future work will rigorously evaluate VACP on non-standard, highly dynamic VA systems to determine its boundary conditions and scalability under greater visual and interactive complexity. ***Visual Nuance:*** Visual nuances like Spatial clustering, Gestalt principles, and dense visual patterns are often lost, when restricting exposure to semantic and structural context. Our experiments indicate that giving agents visual context in addition to the semantic VACP state (L3 and L4), such as interface screenshots, improves run-time and task completion performance, suggesting that multimodal access to layer L1 is crucial when token consumption is not a constraint. ***Adoption Effort and Generalization:*** Implementing VACP adds developer overhead. While our current TypeScript library provides adapters for common declarative frameworks, complex, dynamically reconfigurable and custom VA systems require mapping of application state and interactions to the protocol. Further, the VACP library is designed for web-based systems. Applying VACP beyond web environments will require development of new transport layers and adapters. We envision the core principles of VACP generalizing to data-rich environments like GIS or 3D modeling, as they face, like VA, dynamic app states, available interactions, and precise execution [30].

Research Opportunities — Exposing VA interfaces through a structured, semantic protocol opens several research directions:

Autonomous Agentic Analysis: VACP can enhance autonomous agentic analysis beyond pure data access by providing access to available interactive VA systems. In future, it is relevant to explore how agents adapt their analysis processes given the additional modality. ***Multi-Agent and Mixed-Initiative Systems:*** VACP supports concurrent access to a single application state, enabling mixed-initiative environments where specialized agents (e.g., data-cleaning, or analyst agents) collaborate seamlessly with human users. Future work can examine how this shared state affects user behavior, clarifying the benefits and drawbacks of such collaboration. ***Agent-Driven Co-Adaptive Guidance:*** By enabling intelligent agents to continuously perceive interface state and reason about user interactions, VACP supports a shift from rigid, rule-based guidance systems (e.g., Lotse [59]) to flexible, context-aware assistance tailored to the user's analytical flow. ***Automated Testing of Analytical Provenance:*** Beyond standard functional UI testing, the protocol's structured exposure of system states lets AI agents systematically navigate VA tools, enabling automated testing of complex data mappings and assessing the usability of analytical provenance tracking in highly interactive environments, expanding works such as TraVIS [19]. ***Benchmarking Interactive VA:*** Moving forward, the community would benefit from a standardized benchmark, analogous to VLAT, specifically for assessing AI agent reasoning, interaction performance, and tool use in interactive VA interfaces.

8 Conclusion

We presented VACP, the first context protocol to provide AI agents native access to interactive VA applications, by formalizing minimal requirements for both the interactive VA environment and the agent. Across diverse agent setups and VA tasks, our proof-of-concept VACP implementation significantly outperforms state-of-the-art web-based AI agent interaction methods in reliability and performance. Further, VACP serves as a building block for future mixed-initiative systems, enabling AI agents to seamlessly navigate, interpret, and manipulate VA interfaces alongside human users. To foster its use within the community, our implementation is available as open source software[1].

Acknowledgments

This work was partially funded by Julius Bär and foryouandyourcustomers. The compute credits for the experiments have been provided by Google and Microsoft.

References

[1] Anomaly Innovations Inc. OpenCode: AI coding agent built for the terminal. https://github.com/anomalyco/opencode, 2026. v1.1.53. [Online; accessed 07-January-2026]. 8

[2] Anthropic PBC. Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku. https://www.anthropic.com/news/3-5-models-and-computer-use, 2024. [Online; accessed 09-February-2026]. 1, 2

[3] Anthropic PBC. Claude Sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5, 2025. [Online; accessed 09-February-2026]. 1, 7

[4] Anthropic PBC. Model Context Protocol. https://modelcontextprotocol.io/specification/2025-11-25, 2025. [Online; accessed 12-February-2026]. 2, 5

[5] B. Bach, E. Freeman, A. Abdul-Rahman, C. Turkay, S. Khan, Y. Fan et al. Dashboard Design Patterns. *IEEE Trans. Vis. Comput. Graph.*, 29(1):342–352, 2023. doi: 10.1109/TVCG.2022.3209448 2, 3, 6

[6] L. Battle and J. Heer. Characterizing Exploratory Visual Analysis: A Literature Review and Evaluation of Analytic Provenance in Tableau. *Comput. Graph. Forum*, 38(3):145–159, 2019. doi: 10.1111/CGF.13678 1

[7] A. Bendeck and J. T. Stasko. An Empirical Evaluation of the GPT-4 Multimodal Language Model on Visualization Literacy Tasks. *IEEE Trans. Vis. Comput. Graph.*, 31(1):1105–1115, 2025. doi: 10.1109/TVCG.2024.3456155 2

[8] M. Brehmer and T. Munzner. A Multi-Level Typology of Abstract Visualization Tasks. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2376–2385, 2013. doi: 10.1109/TVCG.2013.124 2, 3, 6, 7

[9] Browser Use Inc. Browser Use: The AI Browser Agent. https://github.com/browser-use/browser-use, 2025. [Online; accessed 09-March-2026]. 1

[10] W. Buxton. Lexical and Pragmatic Considerations of Input Structures. *SIGGRAPH Comput. Graph.*, 17(1):31–37, 1983. doi: 10.1145/988584.988586 6, 8, 9

[11] D. Ceneda, T. Gschwandtner, T. May, S. Miksch, H. Schulz, M. Streit et al. Characterizing Guidance in Visual Analytics. *IEEE Trans. Vis. Comput. Graph.*, 23(1):111–120, 2017. doi: 10.1109/TVCG.2016.2598468 1, 2

[12] K. Choe, C. Lee, S. Lee, J. Song, A. Cho, N. W. Kim et al. Enhancing Data Literacy On-Demand: LLMs as Guides for Novices in Chart Interpretation. *IEEE Trans. Vis. Comput. Graph.*, 31(9):4712–4727, 2025. doi: 10.1109/TVCG.2024.3413195 2

[13] A. K. Das, M. Tarun, and K. Mueller. Charts-of-Thought: Enhancing LLM Visualization Literacy Through Structured Data Extraction. *IEEE Trans. Vis. Comput. Graph.*, 32(1):1427–1437, 2026. doi: 10.1109/TVCG.2025.3634813 2

[14] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang et al. Mind2Web: Towards a Generalist Agent for the Web. In *Advances in Neural Information Processing Systems 36,NeurIPS*, 2023. doi: 10.5555/3666122.3667342 2

[15] V. Dhanoa, A. Wolter, G. M. León, H. Schulz, and N. Elmqvist. Agentic Visualization: Extracting Agent-Based Design Patterns From Visualization Systems. *IEEE Computer Graphics and Applications*, 45(6):89–100, 2025. doi: 10.1109/MCG.2025.3607741 9

[16] V. Dibia. LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics using Large Language Models. In *Proc. Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), ACL*, pp. 113–126. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.ACL-DEMO.11 2

[17] E. Dimara and C. Perin. What is Interaction for Data Visualization? *IEEE Trans. Vis. Comput. Graph.*, 26(1):119–129, 2020. doi: 10.1109/TVCG.2019.2934283 2, 3

[18] F. Elavsky, L. Nadolskis, and D. Moritz. Data Navigator: An Accessibility-Centered Data Navigation Toolkit. *IEEE Trans. Vis. Comput. Graph.*, 30(1):803–813, 2023. doi: 10.1109/TVCG.2023.3327393 2

[19] M. Filosa, A. Plexousaki, M. D. Stadio, F. Bovi, D. Benvenuti, T. Catarci et al. TraVIS: A User Trace Analyzer to Support User-Centered Design of Visual Analytics Solutions. *IEEE Trans. Vis. Comput. Graph.*, 31(10):7614–7631, 2025. doi: 10.1109/TVCG.2025.3546863 9

[20] Gemini Team. Gemini: A Family of Highly Capable Multimodal Models. *CoRR*, abs/2312.11805, 2023. doi: 10.48550/ARXIV.2312.11805 1, 7

[21] C. Gillmann, R. G. C. Maack, F. Raith, J. F. Pérez, and G. Scheuermann. A Taxonomy of Uncertainty Events in Visual Analytics. *IEEE Comput. Graph. and Applications*, 43(5):62–71, 2023. doi: 10.1109/MCG.2023.3299297 2

[22] Google. Computer Use Preview. https://github.com/google-gemini/computer-use-preview, 2026. [Online; accessed 07-January-2026]. 8

[23] J. Gorniak, Y. Kim, D. Wei, and N. W. Kim. VizAbility: Enhancing Chart Accessibility with LLM-based Conversational Interaction. In *Proc. ACM Symp. on User Interface Software and Technology, UIST 2024*, pp. 89:1–89:19. ACM, 2024. doi: 10.1145/3654777.3676414 2

[24] I. Gur, H. Furuta, A. V. Huang, M. Safdari, Y. Matsuo, D. Eck et al. A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis. In *Int. Conf. on Learning Representations, ICLR 2024*. OpenReview.net, 2024. 2

[25] J. Heer and D. Moritz. Mosaic: An Architecture for Scalable & Interoperable Data Views. *IEEE Trans. Vis. Comput. Graph.*, 30(1):436–446, 2024. doi: 10.1109/TVCG.2023.3327189 2, 5

[26] J. Heer, D. Moritz, and R. Pechuk. Mosaic Selections: Managing and Optimizing User Selections for Scalable Data Visualization Systems. *IEEE Trans. Vis. Comput. Graph.*, 32(1):988–998, 2026. doi: 10.1109/TVCG.2025.3634649 2

[27] J. Heer and B. Shneiderman. Interactive Dynamics for Visual Analysis. *Commun. ACM*, 55(4):45–54, 2012. doi: 10.1145/2133806.2133821 2, 3

[28] K. Hong, A. Troynikov, and J. Huber. Context Rot: How Increasing Input Tokens Impacts LLM Performance. https://research.trychroma.com/context-rot, 2025. [Online; accessed 09-February-2026]. 2

[29] James Nurthen, Michael Cooper, Shawn Lawton Henry. WAI-ARIA Overview. https://www.w3.org/WAI/standards-guidelines/aria/, 2025. [Online; accessed 12-March-2026]. 3

[30] J. Jankowski and M. Hachet. A Survey of Interaction Techniques for Interactive 3D Environments. In *Conf. of the European Association for Comput. Graph., Eurographics - State of the Art Reports*, pp. 65–93. Eurographics Association, 2013. doi: 10.2312/CONF/EG2013/STARS/065-093 9

[31] A. Jobst, D. Atzberger, W. Scheibel, M. Tytarenko, J. Döllner, and T. Schreck. Evaluating Large Language Model Integration into Natural Language Interfaces for Visualizations with Available Input Data. In *Proc. Int. Conf. on Comput. Graph., Interaction and Vis. Theory and Applications, GRIVAPP*, pp. 464–475. SCITEPRESS, 2026. doi: 10.5220/0014628600004728 2

[32] S. Jones, I. Pedraza Pineros, D. Hajas, J. Zong, and A. Satyanarayan. "Customization is Key": Reconfigurable Textual Tokens for Accessible Data Visualizations. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 1–14. ACM, Honolulu HI USA, 2024. doi: 10.1145/3613904.3641970 2

[33] A. Kartha, A. Masry, M. S. Islam, T. Lang, S. Rahman, R. Mahbub et al. DashboardQA: Benchmarking Multimodal Agents for Question Answering on Interactive Dashboards. In *Findings of the Association for Computational Linguistics: EACL*, Findings of ACL, pp. 3385–3407. Association for Computational Linguistics, 2026. 1, 2

[34] N. W. Kim, S. C. Joyner, A. Riegelhuth, and Y. Kim. Accessible Visualization: Design Space, Opportunities, and Challenges. *Comput. Graph. Forum*, 40(3):173–188, 2021. doi: 10.1111/CGF.14298 2

[35] S. Lee, S. Kim, and B. C. Kwon. VLAT: Development of a Visualization Literacy Assessment Test. *IEEE Trans. Vis. Comput. Graph.*, 23(1):551–560, 2017. doi: 10.1109/TVCG.2016.2598920 6

[36] L. Y. Lo and H. Qu. How Good (Or Bad) Are LLMs at Detecting Misleading Visualizations? *IEEE Trans. Vis. Comput. Graph.*, 31(1):1116–1125, 2025. doi: 10.1109/TVCG.2024.3456333 2

[37] M. Lu, J. Liang, Y. Zhang, G. Li, S. Chen, Z. Li et al. Interaction+: Interaction Enhancement for Web-based Visualizations. In *IEEE Pacific Visualization Symp., PacificVis 2017*, pp. 61–70. IEEE Computer Society, 2017. doi: 10.1109/PACIFICVIS.2017.8031580 3

[38] A. Lundgard and A. Satyanarayan. Accessible Visualization via Natural Language Descriptions: A Four-Level Model of Semantic Content. *IEEE Trans. Vis. Comput. Graph.*, 28(1):1073–1083, 2022. doi: 10.1109/TVCG.2021.3114770 2

[39] M. Martins, B. Jardim, M. de Castro Neto, and A. Barriguinha. Talking to Data: A Systematic Review of the Rise of Conversational Agents for

Visual Analytics. *IEEE Access*, 13:208902–208931, 2025. doi: 10.1109/ACCESS.2025.3638950 3

[40] S. Monadjemi, M. Guo, D. Gotz, R. Garnett, and A. Ottley. Human-computer collaboration for visual analytics: an agent-based framework. *Comput. Graph. Forum*, 42(3):199–210, 2023. doi: 10.1111/CGF.14823 1, 9

[41] T. Munzner. *Visualization Analysis and Design*. A.K. Peters Visualization Series. A K Peters, 2014. 3, 6

[42] D. A. Norman. *The Design of Everyday Things*. Basic Books, Inc., USA, 2002. 6, 8

[43] C. North, R. Chang, A. Endert, W. Dou, R. May, B. Pike et al. Analytic Provenance: Process + Interaction + Insight. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2011, Extended Abstracts Volume*, pp. 33–36. ACM, 2011. doi: 10.1145/1979742.1979570 2

[44] OpenAI. Computer-Using Agent: Introducing a universal interface for AI to interact with the digital world. https://openai.com/index/computer-using-agent, 2025. [Online; accessed 09-March-2026]. 1, 2

[45] OpenAI. GPT-5.2. https://openai.com/index/introducing-gpt-5-2/, 2025. [Online; accessed 09-February-2026]. 1, 7

[46] S. Pandey and A. Ottley. Benchmarking visual language models on standardized visualization literacy tests. *Comput. Graph. Forum*, 44(3), 2025. doi: 10.1111/CGF.70137 2

[47] W. A. Pike, J. Stasko, R. Chang, and T. A. O'Connell. The Science of Interaction. *Information Visualization*, 8(4):263–274, 2009. doi: 10.1057/ivs.2009.22 2, 3

[48] P. Pirolli and S. Card. The Sensemaking Process and Leverage Points for Analyst Technology as Identified Through Cognitive Task Analysis. In *Proc. Int. Conf. on Intelligence Analysis*, vol. 5, pp. 2–4, 2005. 6, 8

[49] J. R. Quinlan. Combining Instance-Based and Model-Based Learning. In *Machine Learning, Proc. Int. Conf.*, pp. 236–243. Morgan Kaufmann, 1993. doi: 10.1016/B978-1-55860-307-3.50037-X 7

[50] D. Ren, F. Hohman, H. Lin, and D. Moritz. Embedding Atlas: Low-Friction, Interactive Embedding Visualization. In *2025 IEEE Visualization and Visual Analytics (VIS)*, pp. 191–195, 2025. doi: 10.1109/VIS60296.2025.00044 2

[51] D. Sacha, H. Senaratne, B. C. Kwon, G. P. Ellis, and D. A. Keim. The Role of Uncertainty, Awareness, and Trust in Visual Analytics. *IEEE Trans. Vis. Comput. Graph.*, 22(1):240–249, 2016. doi: 10.1109/TVCG.2015.2467591 2

[52] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Trans. Vis. Comput. Graph.*, 23(1):341–350, 2017. doi: 10.1109/TVCG.2016.2599030 2, 5

[53] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Trans. Vis. Comput. Graph.*, 22(1):659–668, 2016. doi: 10.1109/TVCG.2015.2467091 2

[54] T. Schick, J. Dwivedi-Yu, R. Dessí, R. Raileanu, M. Lomeli, E. Hambro et al. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Proc. Int. Conf. on Neural Information Processing Systems*, NIPS '23, art. no. 2997, 13 pp. Curran Associates Inc., 2023. 2

[55] K. Sedig and P. Parsons. Interaction Design for Complex Cognitive Activities with Visual Representations: A Pattern-Based Approach. *AIS Trans. on Human-Computer Interactions*, 5(2):84–133, 2013. 2

[56] Selenium Project. Selenium. https://www.selenium.dev/. [Online; accessed 11-February-2026]. 2

[57] P. Shaw, M. Joshi, J. Cohan, J. Berant, P. Pasupat, H. Hu et al. From Pixels to UI Actions: Learning to Follow Instructions via Graphical User Interfaces. In *Advances in Neural Information Processing Systems 36, NeurIPS*, 2023. 2

[58] T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang. World of Bits: An Open-Domain Platform for Web-Based Agents. In *Proc. Int. Conf. on Machine Learning, ICML 2017*, vol. 70, pp. 3135–3144. PMLR, 2017. 2

[59] F. Sperrle, D. Ceneda, and M. El-Assady. Lotse: A Practical Framework for Guidance in Visual Analytics. *IEEE Trans. Vis. Comput. Graph.*, 29(1):1124–1134, 2023. doi: 10.1109/TVCG.2022.3209393 9

[60] F. Sperrle, A. Jeitler, J. Bernard, D. A. Keim, and M. El-Assady. Co-Adaptive Visual Data Analysis and Guidance Processes. *Comput. Graph.*, 100:93–105, 2021. doi: 10.1016/J.CAG.2021.06.016 2

[61] F. Sperrle, H. Schäfer, D. Keim, and M. El-Assady. Learning Contextualized User Preferences for Co-Adaptive Guidance in Mixed-Initiative Topic Model Refinement. *Computer Graphics Forum*, 40(3):215–226, 2021. doi: 10.1111/cgf.14301 1

[62] T. Stähle, M. J. op de Haar, S. Boyer, R. Sevastjanova, A. Narechania, and M. El-Assady. A Design Space for Intelligent Agents in Mixed-Initiative Visual Analytics. *CoRR*, abs/2512.23372, 2025. doi: 10.48550/ARXIV.2512.23372 1

[63] M. Valentim, V. Dhanoa, G. M. León, and N. Elmqvist. The Plot Thickens: Quantitative Part-by-Part Exploration of MLLM Visualization Literacy. *CoRR*, abs/2504.02217, 2025. doi: 10.48550/ARXIV.2504.02217 2

[64] B. Walderman, L. Lee, A. Nolan, D. Bokan, K. Sagar, and H. V. Opstal. WebMCP. https://github.com/webmachinelearning/webmcp, 2026. [Online; accessed 11-March-2026]. 2

[65] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang et al. A Survey on Large Language Model based Autonomous Agents. *Frontiers Comput. Sci.*, 18(6):186345, 2024. doi: 10.1007/S11704-024-40231-1 1, 2

[66] Y. Wang, M. Li, and H. Chen. A Case for Declarative LLM-friendly Interfaces for Improved Efficiency of Computer-Use Agents. *CoRR*, abs/2510.04607, 2025. doi: 10.48550/ARXIV.2510.04607 3

[67] WHATWG community. DOM – Living Standard. https://dom.spec.whatwg.org, 2026. [Online; accessed 12-March-2026]. 2

[68] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan et al. ReAct: Synergizing Reasoning and Acting in Language Models. In *Int. Conf. on Learning Representations, ICLR 2023*. OpenReview.net, 2023. 7

[69] J. S. Yi, Y. ah Kang, J. T. Stasko, and J. A. Jacko. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1224–1231, 2007. doi: 10.1109/TVCG.2007.70515 2

[70] A. L. Zhang, T. Kraska, and O. Khattab. Recursive Language Models. *CoRR*, abs/2512.24601, 2025. doi: 10.48550/ARXIV.2512.24601 2

[71] Y. Zhao, Y. Wang, X. Luo, Y. Wang, and J. Fekete. Libra: An Interaction Model for Data Visualization. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2025*, pp. 1169:1–1169:17. ACM, 2025. doi: 10.1145/3706598.3713769 2, 3

[72] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *Int. Conf. on Learning Representations, ICLR 2024*. OpenReview.net, 2024. 2

[73] J. Zong, C. Lee, A. Lundgard, J. Jang, D. Hajas, and A. Satyanarayan. Rich Screen Reader Experiences for Accessible Data Visualization. *Comput. Graph. Forum*, 41(3):15–27, 2022. doi: 10.1111/CGF.14519 2