MultiVis-Agent: A Multi-Agent Framework with Logic Rules for Reliable and Comprehensive Cross-Modal Data Visualization

JINWEI LU, The Hong Kong Polytechnic University, China
YUANFENG SONG, ByteDance, China
CHEN ZHANG, The Hong Kong Polytechnic University, China
RAYMOND CHI-WING WONG, The Hong Kong University of Science and Technology, China

Real-world visualization tasks involve complex, multi-modal requirements that extend beyond simple text-to-chart generation, requiring reference images, code examples, and iterative refinement. Current systems exhibit fundamental limitations: single-modality input, one-shot generation, and rigid workflows. While LLM-based approaches show potential for these complex requirements, they introduce reliability challenges including catastrophic failures and infinite loop susceptibility. To address this gap, we propose **MultiVis-Agent**, a logic rule-enhanced multi-agent framework for reliable multi-modal and multi-scenario visualization generation. Our approach introduces a four-layer logic rule framework that provides mathematical guarantees for system reliability while maintaining flexibility. Unlike traditional rule-based systems, our logic rules are mathematical constraints that guide LLM reasoning rather than replacing it. We formalize the **MultiVis** task spanning four scenarios from basic generation to iterative refinement, and develop **MultiVis-Bench**, a benchmark with over 1,000 cases for multi-modal visualization evaluation. Extensive experiments demonstrate that our approach achieves 75.63% visualization score on challenging tasks, significantly outperforming baselines (57.54-62.79%), with task completion rates of 99.58% and code execution success rates of 94.56% (vs. 74.48% and 65.10% without logic rules), successfully addressing both complexity and reliability challenges in automated visualization generation.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**.

Additional Key Words and Phrases: Multi-Agent Framework, Natural Language to SQL, Data Visualization, Database Applications, Automated Code Generation

ACM Reference Format:
Jinwei Lu, Yuanfeng Song, Chen Zhang, and Raymond Chi-Wing Wong. 2026. MultiVis-Agent: A Multi-Agent Framework with Logic Rules for Reliable and Comprehensive Cross-Modal Data Visualization. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 56 (February 2026), 25 pages. https://doi.org/10.1145/3786670

1 Introduction

Despite extensive research in automated visualization systems within the database and data mining community [14, 16, 22, 25, 28, 32, 40], existing approaches face fundamental limitations that prevent practical deployment. Current database-centric systems focus on single-modal text-to-visualization translation, lacking multi-modal capabilities—the ability to process diverse inputs including reference images, reference code, and existing visualizations beyond natural language—and iterative

Authors' Contact Information: Jinwei Lu, jwlu18@gmail.com, The Hong Kong Polytechnic University, Hong Kong SAR, China; Yuanfeng Song, songyf@outlook.com, ByteDance, China; Chen Zhang, jason-c.zhang@polyu.edu.hk, The Hong Kong Polytechnic University, Hong Kong SAR, China; Raymond Chi-Wing Wong, raywong@cse.ust.hk, The Hong Kong University of Science and Technology, Hong Kong SAR, China.

ACM 2836-6573/2026/2-ART56
https://doi.org/10.1145/3786670

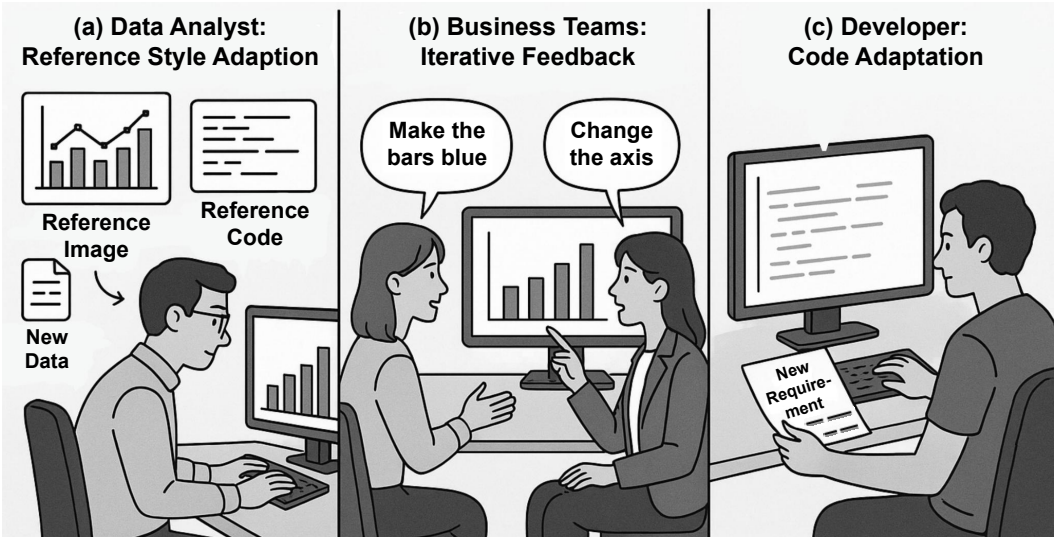


Fig. 1. Real-world visualization tasks require multi-modal inputs and iterative refinement. Current Text-to-Vis systems fail to support these scenarios.

refinement mechanisms essential for real-world analytical workflows. These systems suffer from critical reliability issues: LLM-based agents produce inconsistent outputs, while traditional pipelines are too rigid for complex scenarios involving reference images, code, or iterative feedback.

System Limitations. Real-world visualization tasks extend far beyond simple text-to-vis generation (Figure 1). Analysts must adapt reference styles to new datasets, teams require iterative refinement based on feedback, and developers need to modify existing code for new requirements. Despite progress in Text-to-Visualization research [4, 15, 18, 20, 22–24, 29–32, 36, 37, 39, 44, 45], current systems exhibit three fundamental limitations: (1) **single-modality input**—systems like Chat2VIS [24] cannot interpret reference images or code examples; (2) **one-shot generation**—approaches like Prompt4Vis [18] employ single-pass processes unsuitable for iterative refinement; and (3) **rigid workflows**—systems like nvAgent [26] use fixed processes that cannot adapt to diverse task types.

The Reliability Crisis. LLM-based agent systems have introduced critical reliability challenges: (1) **catastrophic failures** with infinite error loops; (2) **uncontrolled error propagation** through entire workflows; (3) **parameter boundary violations** causing execution failures; and (4) **infinite loop susceptibility** in iterative processes lacking termination guarantees.

Our Solution: MultiVis-Agent. To address these challenges, we propose **MultiVis-Agent**, a novel logic rule-enhanced multi-agent framework for reliable multi-modal visualization generation. Our approach decomposes complex visualization tasks into specialized sub-problems handled by coordinated agents. The centralized architecture enables consistent state management, global context awareness, and effective integration of diverse inputs. At its core, a **Coordinator Agent** dynamically orchestrates specialized agents for database interaction, code generation, and validation based on task context, enabling effective multi-modal fusion and state-aware iterative refinement.

Key Innovations. We introduce a four-layer logic rule framework that provides mathematical guarantees for system reliability: (1) parameter boundary constraints, (2) deterministic task classification, (3) systematic error recovery, and (4) guaranteed loop termination. These formal constraints

Table 1. Comparison of MultiVis-Bench with existing visualization benchmarks.

Datasets	#-Tables	#-Samples	Ref. Image Input	Ref. Code Input	Output Format	Scenario Types*	Construction Leader**
Quda [7]	36	14035	✗	✗	Analytic Tasks	ATR	Human
NLV Corpus [34]	3	814	✗	✗	Vega-Lite	BG	Human
Dial-NVBench [31]	780	124,449	✗	✗	VQL	BG	Program
VL2NL [15]	1,981	3,962	✗	✗	Vega-Lite	BG	LLM
VisEval [1]	748	2,524	✗	✗	VQL	BG	LLM
nvBench [22]	780	25,750	✗	✗	VQL	BG	Program
nvBench 2.0 [21]	780	24,076	✗	✗	VQL	BG	LLM
MultiVis-Bench	697	1,202	✓	✓	Python Code	BG, IRG, CRG, IR	Human+LLM

* **Scenario Types:** ATR: Analytic Task Recognition; BG: Basic Generation; IRG: Image-Referenced Generation; CRG: Code-Referenced Generation; IR: Iterative Refinement.

** **Construction Leader:** Human: Expert-authored; Program: logic rule-based generation; LLM: AI-generated with minimal oversight; Human+LLM: Human-led with AI assistance and expert validation.

guide and constrain LLM-driven decisions while maintaining flexibility for complex visualization tasks.

Contributions. Our work has four main contributions:

- **Logic Rule-Enhanced Multi-Agent Architecture:** We propose MultiVis-Agent [1], a novel framework with a four-layer logic rule system providing mathematical constraints for parameter safety, error recovery, and termination guarantees. This achieves task completion rates of 98.68-100% and code execution success rates of 94.56-97.10%, dramatically outperforming the same framework without logic rules (74.48-92.03% and 63.18-83.33% respectively).
- **MultiVis Task Formulation:** We formalize MultiVis, extending traditional Text-to-Vis to four scenarios: Basic Generation, Image-Referenced Generation, Code-Referenced Generation, and Iterative Refinement, systematically capturing real-world multi-modal visualization requirements.
- **MultiVis-Bench Benchmark:** We construct a novel benchmark with over 1,000 cases supporting multi-modal inputs and executable Python code output, enabling end-to-end evaluation unlike existing datasets with single-modal inputs and intermediate representations.
- **Superior Empirical Performance:** Extensive experiments show substantial improvements, with MultiVis-Agent achieving 75.63% visualization quality in challenging Image-Referenced Generation versus 62.79% (LLM Workflow) and 57.54% (Instructing LLM). Performance remains strong across all scenarios (72.99-76.58%), with logic rules contributing 17.58-31.70pp improvements.

The rest of this paper is organized as follows: Section 2 introduces the MultiVis task formulation. Section 3 describes the MultiVis-Bench dataset. Section 4 details the MultiVis-Agent framework. Section 5 explains our evaluation methodology. Section 6 presents the experimental results. Section 7 reviews related work, and Section 8 concludes the paper.

2 MultiVis: A Comprehensive Task for Visualization Generation

2.1 Overview and Motivation

The prevailing Text-to-Visualization (Text-to-Vis) paradigm, while foundational, often struggles with the complexity and dynamism inherent in practical data analysis workflows. Its typical focus on single-turn, text-only requests does not fully capture the multi-modal nature of real-world inputs (e.g., reference images and code examples) or the essential iterative refinement process common in visualization design. To systematically address these challenges and establish a more encompassing

[1]The source code and data are available at https://github.com/Jinwei-Lu/MultiVis.git.

framework, we propose **MultiVis**, a comprehensive task formulation that significantly extends traditional Text-to-Vis by unifying diverse generation and refinement scenarios.

The MultiVis task formulation offers several key advantages: (i) it explicitly supports multimodal inputs, reflecting real user requirements; (ii) it systematically covers key stages of the visualization *generation and refinement* lifecycle; (iii) its structured categorization provides a clear framework for addressing diverse scenarios; and (iv) its progression from basic to complex situations facilitates systematic system development and evaluation. This comprehensive formulation motivates the need for flexible and adaptive system architectures, such as the agent-based approach detailed in Section 4, capable of handling the distinct requirements of each MultiVis scenario.

2.2 Task Formulation

Formally, we define MultiVis as a family of tasks aimed at learning functions that transform various combinations of inputs into executable visualization code. Let D represent a database, Q a natural language query (NLQ), I_{ref} a reference image, C_{ref} reference code, V_{old} existing visualization code to be refined, and V the target visualization code (either newly generated or refined). The core task within each MultiVis scenario is to learn a mapping $f : \mathcal{X} \rightarrow V$, where $\mathcal{X}$ is a specific combination of input modalities (such as Q, D, I_{ref}, C_{ref} and V_{old}). Based on the composition of these inputs and the specific goal, we categorize MultiVis into four distinct scenarios:

- **Scenario A (BG, Basic Generation):** Inputs $\mathcal{X} = (Q, D)$; Output target is V. This corresponds to traditional Text-to-Vis. *Example:* Q: "Show sales trends by month"; D: Table [month, sales_amount] $\rightarrow V$: `alt.Chart(data).mark_line().encode(x='month:T', y='sales_amount:Q')`. *Existing systems:* Traditional approaches handle straightforward scenarios but struggle with ambiguous queries.
- **Scenario B (IRG, Image-Referenced Generation):** Inputs $\mathcal{X} = (Q, D, I_{ref})$; Output target is V. The image I_{ref} provides style, layout, or chart type cues, introducing *cross-modal understanding* challenges. *Example:* Q: "Create a visualization like this reference image"; D: Table [sales, category, region]; I_{ref}: Stacked bar chart with custom colors $\rightarrow V$: `alt.Chart(data).mark_bar().encode(x='sum(sales):Q', y='category:N', color='region:N')`. *Existing systems:* Text-only approaches cannot interpret visual style cues, leading to generic visualizations.
- **Scenario C (CRG, Code-Referenced Generation):** Inputs $\mathcal{X} = (Q, D, C_{ref})$; Output target is V. Reference code C_{ref} provides implementation patterns, posing *code understanding and adaptation* challenges. *Example:* Q: "Adapt this matplotlib code using Altair and current data"; D: Table [category]; C_{ref}: `plt.scatter(x, y, c=colors, alpha=0.6)` $\rightarrow V$: `alt.Chart(data).mark_circle(opacity=0.6).encode(x= 'x:Q',y='y:Q',color='category: N')`. *Existing systems:* Current approaches struggle with cross-library translation and semantic code understanding.
- **Scenario D (IR, Iterative Refinement):** Inputs $\mathcal{X} = (Q, D, V_{old})$; Output target is V (refined code). The existing code V_{old} is modified based on feedback, requiring *dialogue state tracking and precise modification. Example:* Q: "Make x-axis labels vertical and add title"; D: Table [month]; V_{old}: `alt.Chart(data).mark_bar(). encode(x='month:N', y='sales:Q')` $\rightarrow V$: `...encode(x=alt. X('month:N', axis=alt.Axis(labelAngle=-90)), y='sales:Q').properties(title='Sales by Month')`. *Existing systems:* One-shot approaches cannot handle iterative feedback loops.

For each scenario $S \in \{A, B, C, D\}$, this mapping is learned by a specialized function $f_S(\cdot) \in \mathcal{F}_{\theta_S}$, parameterized by θ_S. This explicit distinction underscores that effectively handling the unique input combinations and inherent challenges of each scenario (e.g., interpreting image styles, adapting code structures, and implementing precise modifications) likely requires different underlying model capabilities or architectural components, moving beyond a one-size-fits-all approach.

3 MultiVis-Bench: Benchmark of MultiVis

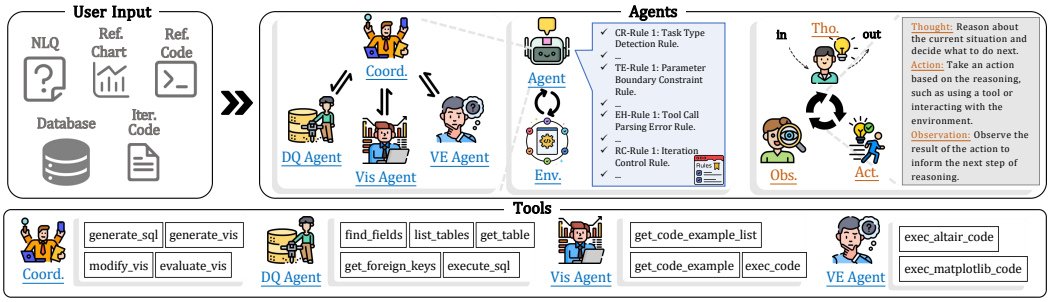


Fig. 2. The architecture of MultiVis-Agent. The framework processes multi-modal user inputs (NLQ, Reference Chart, Reference Code, Iterative Code, Database) through a central Coordinator Agent that dynamically orchestrates specialized agents: the Database & Query Agent (DQ Agent), the Visualization Implementation Agent (Vis Agent), and the Validation & Evaluation Agent (VE Agent). Each agent is equipped with specific tools and follows a reasoning cycle (Thought-Observation-Action). The system is governed by a four-layer logic rule framework (CR, TE, EH, RC logic rules) that ensures reliable execution and error handling, enabling task-adaptive visualization generation and iterative refinement.

3.1 Overview and Motivation

To evaluate systems designed for complex, multi-modal, and iterative visualization tasks, which existing benchmarks often inadequately cover, we introduce **MultiVis-Bench**. It is a novel benchmark specifically constructed to assess capabilities across the comprehensive MultiVis scenarios (A-D) defined in Section 2. MultiVis-Bench focuses on crucial capabilities such as handling diverse **multi-modal inputs** (images, code), supporting **iterative refinement** processes, and generating directly **executable visualization code**. These aspects are critical for advanced systems like MultiVis-Agent but are poorly covered by benchmarks limited to basic Text-to-Vis or intermediate, non-executable representations.

Table 1 compares MultiVis-Bench with existing visualization benchmarks, contextualizing its contributions and highlighting its unique features, including multi-modal input support, executable Python code output, and full lifecycle scenario coverage.

3.2 Dataset Construction Methodology

MultiVis-Bench was constructed through a systematic, human-led approach with LLM assistance, focusing on quality, diversity, and representativeness across the four MultiVis scenarios.

Foundational Resources. We prepared two essential resource types: (1) *Databases*: 141 complex SQLite databases (average 5.8 tables, 27 columns) selected from the Spider benchmark [43], filtered for visualization suitability based on predefined criteria (presence of numerical/categorical data, sufficient data volume and domain diversity) across multiple domains including business, science, and social studies; (2) *Visualization Templates*: Standardized Altair code templates for 127 distinct chart types (bar charts, scatter plots, area charts, composite charts, etc.) sourced from official Altair documentation and curated visualization examples.

Scenario-Specific Construction. For each MultiVis scenario, we employed a human-led, LLM-assisted approach where initial candidates were generated by an LLM (Gemini-2.0-pro-exp) prompted

with database schemas, visualization templates, and scenario-specific instructions, then underwent rigorous human expert review and refinement (averaging 2.5 rounds per example).

For **Scenario A (BG)**, we provided Altair documentation and database to generate candidate pairs of (NLQ, visualization code), with human experts selecting or manually adjusting examples to ensure quality, resulting in 306 examples covering 127 chart types. **Scenario B (IRG)** involved human experts first curating 109 reference chart images (covering 107 chart types), then manually rephrasing and adapting queries from Scenario A to incorporate visual cues from reference images, instructing systems to generate visualizations matching reference styles with new data. **Scenario C (CRG)** utilized 132 Altair and 101 Matplotlib reference code snippets manually curated by experts, who adapted Scenario A queries to use these external examples as structural/stylistic guides, totaling 233 examples. **Scenario D (IR)** built upon Scenario A cases, where experts formulated follow-up modification instructions and crafted correctly modified code to simulate realistic iterative refinement processes, resulting in 554 examples.

Quality Assurance. All examples underwent comprehensive validation ensuring: (1) **Technical Correctness** (executable, error-free code producing intended chart structures), (2) **Semantic Faithfulness** (accurate query-visualization alignment and reference adherence), and (3) **Perceptual Effectiveness** (clear, interpretable charts following visualization best practices). The human review process utilized detailed checklists covering code style, requirement adherence, data mapping accuracy, visual clarity, and chart type appropriateness. Additional automated quality control included: (i) code formatting standardization using tools like Black, and (ii) automated execution validation of all code snippets against their respective databases. Substandard examples were iteratively corrected or rejected if fundamental flaws persisted. The complete construction involved iterative LLM-assisted generation with rigorous human expert validation (averaging 2.5 review rounds per example), ensuring high-quality examples across all technical, semantic, and perceptual dimensions.

3.3 Dataset Statistics and Characteristics

MultiVis-Bench contains 1,202 carefully selected cases comprising four task types corresponding to the scenarios in Section 2: Scenario A (BG) with 306 examples; Scenario B (IRG) with 109 examples; Scenario C (CRG) with 233 examples; and Scenario D (IR) with 554 examples. The dataset covers 127 chart types and utilizes 141 complex SQLite databases, allowing targeted evaluation across different visualization stages with progressively complex input modalities beyond traditional text+database input.

As detailed in Table 1, MultiVis-Bench's key distinctions include native support for **multi-modal inputs** (text, database, image, code), generating directly executable **Python code**, a hybrid **construction approach** (human-led with LLM assistance) balancing quality and scale, and comprehensive **scenario coverage** (Basic Generation to Iterative Refinement) for evaluating systems across the entire visualization lifecycle. Unlike previous benchmarks focusing on intermediate, non-executable representations (e.g., VQL and Vega-Lite) from text-only inputs, MultiVis-Bench produces directly executable Altair Python code, facilitating end-to-end evaluation and accommodating multi-modal inputs to reflect more realistic visualization workflows.

4 MultiVis-Agent: System Implementation

This section details the design and implementation of **MultiVis-Agent**, our logic rule-enhanced multi-agent collaborative framework for multi-modal and multi-scenario visualization generation and refinement. We first provide an overview of our design philosophy and system architecture

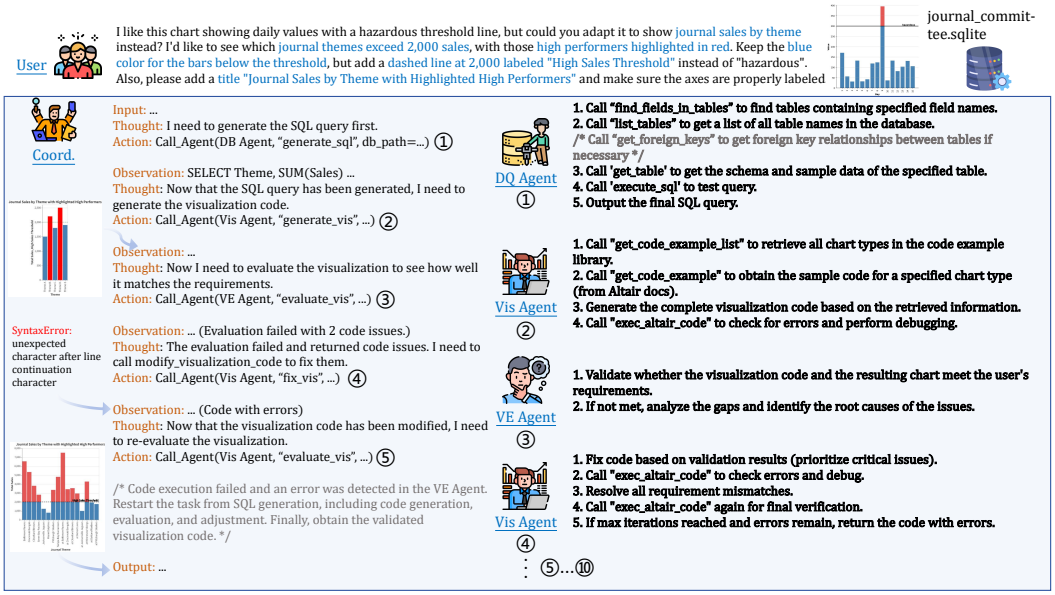


Fig. 3. An example for the working process of MultiVis-Agent.

(Section 4.1). We then illustrate the framework's operational workflow through a concrete execution example (Section 4.2). Next, we present our core innovation: the logic rule-enhanced agent architecture that ensures reliable behavior while maintaining flexibility for complex visualization tasks (Section 4.3). Finally, we describe the multi-agent coordination framework that orchestrates specialized agents for complex visualization tasks (Section 4.4).

4.1 Design Philosophy and Architecture Overview

Traditional visualization generation systems face fundamental limitations that hinder their practical deployment. Pure LLM-based agents exhibit inherent instability, producing inconsistent outputs and lacking robust error handling mechanisms. Traditional pipeline-based methods, while predictable, are too rigid for complex multi-modal scenarios. To address these challenges, we propose a novel approach that combines the reliability of logic rule-enhanced behavior with the flexibility of multi-agent coordination.

4.1.1 Architecture Overview. MultiVis-Agent employs a centralized multi-agent architecture that provides both specialization and flexibility. As depicted in Figure 2, the framework decomposes complex visualization tasks into sub-problems handled by specialized agents (Database & Query, Visualization Implementation, Validation & Evaluation), while a central Coordinator manages the overall process through logic rule-enhanced decision making.

The centralized approach is particularly advantageous for visualization tasks because it enables: (1) maintaining global consistency across visual elements during generation and refinement, (2) integrating holistic feedback from validation processes, (3) facilitating robust state management and history tracking for effective iterative refinement, and (4) providing better error isolation and recovery mechanisms through our logic rule framework.

4.2 Concrete Execution Example

This subsection provides an overview of the technical implementation of MultiVis-Agent. Figure 3 illustrates the framework's operational workflow, showcasing how the Coordinator Agent orchestrates specialized agents to handle user requests through cycles of data retrieval, visualization generation, evaluation, and refinement. MultiVis-Agent employs three specialized agents, each with distinct responsibilities:

(i) *Database & Query Agent.* Understands data requirements and retrieves suitable data through five core tools: `list_tables`, `get_table`, `get_foreign_keys`, `find_fields`, and `execute_sql`. The main `generate_sql` interface (as illustrated in Figure 3①) performs iterative exploration to formulate correct SQL queries.

(ii) *Visualization Implementation Agent.* Translates data and requirements into executable Altair code using three tools: `get_code_ example_list`, `get_code_example`, and `execute_code`. Provides two interfaces: `generate_visualization_code` for initial generation (as illustrated in Figure 3②) and `modify_visualization_code` (demonstrated in Figure 3④) for iterative refinement.

(iii) *Validation & Evaluation Agent.* Assesses technical correctness and perceptual effectiveness using two tools: `execute_altair _code` and `execute_matplotlib_code`. The `evaluate_visualization` interface (illustrated in Figure 3③ and ⑤) performs structured assessment and provides actionable feedback for refinement.

4.3 Logic rule-enhanced Agent Architecture

Our core innovation lies in the logic rule-enhanced agent architecture that ensures reliable behavior while maintaining flexibility for complex visualization tasks. In the formal definitions that follow, we use $\mathcal{F}_S(\cdot) \in \mathcal{F}_{\theta_S}$ to denote the learned mapping function for scenario S parameterized by θ_S, $\mathcal{T}_{tool}$ for the set of available tools, $\mathcal{R} = \{CR, TE, EH, RC\}$ for our four-layer logic rule framework, predicates like $\mathcal{V}(t, s)$ for validation functions, and $T_{max} = 10$ for the maximum iteration bound. We establish the design principles, introduce the four-layer logic rule framework, and provide theoretical guarantees.

4.3.1 Logic rule-enhanced Design Principles. Our key insight is that **mathematical constraints** can guide and constrain LLM-driven decisions while preserving adaptability, fundamentally addressing the system robustness crisis. We introduce *logic rule-enhanced agents*—systems where agent behavior is constrained by formally defined logic rules that ensure graceful error recovery and prevent system crashes. Unlike traditional rule-based systems that rely on static if-then rules, our logic rules are mathematically formalized constraints that provide formal guarantees while maintaining the flexibility of LLM reasoning.

Our logic rule-enhanced architecture is built on four fundamental principles: (1) **System Robustness**: Prevent catastrophic failures and enable graceful degradation under error conditions; (2) **Error Recovery Mechanisms**: Implement structured error detection, classification, and targeted recovery strategies; (3) **Execution Safety**: Ensure safe tool invocations through parameter validation and boundary enforcement; and (4) **Termination Guarante**: Provide mathematical assurance of finite execution time and loop prevention.

4.3.2 Four-Layer logic rule Framework. Our logic rule framework consists of four hierarchical layers that define explicit behavioral constraints through formal mathematical specifications. These layers are applied hierarchically across our agent architecture: CR rules govern the Coordinator Agent's task classification and decision logic; TE and EH rules constrain all four agents' tool execution and error handling; RC rules manage the entire system's iteration control and termination.

Coordination logic rules (CR): System-Level Decision Making.
(i) CR-Rule 1: Task Type Detection. Define task classification function $\mathcal{F}_T : \mathcal{I} \to \mathcal{T}_{task}$ that maps input domain $\mathcal{I}$ to task types $\mathcal{T}_{task} = \{A, B, C, D\}$ with priority ordering $D \succ C \succ B \succ A$:

$$\mathcal{F}_T(I) = \arg\max_{t \in \mathcal{T}_{task}} \pi(t, I) \tag{1}$$

where $I \in \mathcal{I}$ is an input instance, $t \in \mathcal{T}_{task}$ is a task type, and $\pi : \mathcal{T}_{task} \times \mathcal{I} \to \mathbb{R}$ is the priority function. This ensures deterministic task identification for existing code (D), Python reference (C), image reference (B), or basic text (A).

The priority ordering $D \succ C \succ B \succ A$ distinguishes input explicitness to ensure complete task coverage: existing code (D) enables precise modification; code references (C) provide explicit structural guidance; image references (B) require expensive cross-modal understanding; basic text (A) starts from scratch. When multiple input types coexist (e.g., both image and code), the system classifies the task by the highest-priority category, guaranteeing deterministic routing and optimal resource utilization.
(ii) CR-Rule 2: Tool Prerequisite Validation. Let $\mathcal{T}_{tool}$ denote the set of available tools and $\mathcal{S}$ the set of all possible system states. Define validation predicate $\mathcal{V} : \mathcal{T}_{tool} \times \mathcal{S} \to \{0, 1\}$ that ensures tool execution safety:

$$\mathcal{V}(t, s) = \mathbf{1}[\mathcal{P}(t) \subseteq \mathcal{D}(s)] \tag{2}$$

where t is a tool, s is system state, $\mathcal{P}(t)$ denotes prerequisite set for tool t, $\mathcal{D}(s)$ denotes defined attributes in state s, and $\mathbf{1}[\cdot]$ is the indicator function.
(iii) CR-Rule 3: Evaluation Response Decision. Let $\mathcal{E}_{result}$ be the set of possible evaluation results and $\mathcal{A}_{action}$ the set of possible system actions. Define decision function $\mathcal{N} : \mathcal{E}_{result} \to \mathcal{A}_{action}$ that maps evaluation results to system actions:

$$\mathcal{N}(e) = f_d(e.passed, |e.recommendations|) \tag{3}$$

where $e \in \mathcal{E}_{result}$ is evaluation result, f_d is deterministic mapping function, $e.passed$ indicates success, and $|e.recommendations|$ is recommendation count.

For example, if validation passes without recommendations, the decision function returns "task complete." If validation fails with some critical issues, it returns "modify with prioritized feedback." This deterministic mapping ensures consistent system behavior.

Tool Execution logic rules (TE): Parameter Safety and Environment Control.
(i) TE-Rule 1: Parameter Boundary Constraint. Let $\mathcal{P}$ denote the set of all tool parameters. Define constraint function $\phi : \mathcal{P} \times \mathbb{R} \to \mathbb{R} \cup \{\perp\}$ that enforces safety bounds:

$$\phi(p, v) = \begin{cases} \max(\min(v, u_p), l_p) & \text{if } v \in \mathcal{V}_p \\ \perp & \text{otherwise} \end{cases} \tag{4}$$

where $p \in \mathcal{P}$ is parameter, v is its value, $[l_p, u_p]$ is valid range, $\mathcal{V}_p$ is validity domain, and $\perp$ denotes invalid outcome.
(ii) TE-Rule 2: Code Execution Environment. Let C be the set of all possible input code snippets and C' the set of all possible standardized code snippets. Define standardization function $\psi : C \to C'$ that ensures execution consistency:

$$\psi(c) = \mathcal{N}(c) \cup \mathcal{M}(c) \cup \mathcal{I}(c) \tag{5}$$

where $c \in C$ is code snippet, $\mathcal{N}(c)$ performs namespace standardization, $\mathcal{M}(c)$ applies pattern matching, and $\mathcal{I}(c)$ injects save operations.

(iii) TE-Rule 3: Reference Material Processing. Let $\mathcal{R}_{type}$ be the set of reference material types, $\mathcal{F}_{path}$ the set of possible file paths, and $\mathcal{P}_{ref}$ the set of processed reference materials. Define processing function $\rho : \mathcal{R}_{type} \times \mathcal{F}_{path} \to \mathcal{P}_{ref}$ that enforces reference utilization:

$$\rho(t, f) = \mathcal{T}_{proc}(t) \circ \mathcal{V}_{file}(f) \circ \mathcal{P}_{proc}(t, f) \tag{6}$$

where $t \in \mathcal{R}_{type}$ is reference type, $f \in \mathcal{F}_{path}$ is file path, $\mathcal{T}_{proc}(t)$ determines processing type, $\mathcal{V}_{file}(f)$ validates file accessibility, $\mathcal{P}_{proc}(t, f)$ applies type-specific processing, and $\circ$ denotes function composition.

Error Handling logic rules (EH): Systematic Error Recovery. Let $\mathcal{E}_{error}$ denote the set of all possible errors, $C_{context}$ the set of possible execution contexts, and $\mathcal{R}_{recovery}$ the set of possible recovery actions.

(i) EH-Rule 1: Tool Call Parsing Error. Let $\mathcal{T}_{text}$ be the set of all possible text outputs from LLMs and $\mathcal{E}_{type}$ the set of specific error types. Define error classification function $\epsilon : \mathcal{T}_{text} \to \mathcal{E}_{type}$:

$$\epsilon(x) = C(\mathcal{S}(x), \mathcal{T}_{struct}(x), O(x)) \tag{7}$$

where $x \in \mathcal{T}_{text}$ is input text, $\mathcal{S}(x)$, $\mathcal{T}_{struct}(x)$, $O(x)$ analyze syntax, structure, and content respectively, and C performs classification.

(ii) EH-Rule 2: Code Execution Error Recovery. Define recovery function $\delta : \mathcal{E}_{error} \times C_{context} \to \mathcal{R}_{recovery}$:

$$\delta(e, c) = \mathcal{R}_{\tau(e)}(e, c) \tag{8}$$

where $e \in \mathcal{E}_{error}$ is error instance, $c \in C_{context}$ is context, $\tau(e)$ determines error type.

(iii) EH-Rule 3: Image Processing Validation. Let $\mathcal{I}_{path}$ be the set of possible image paths. Define validation predicate $\nu : \mathcal{I}_{path} \to \{0, 1\}$:

$$\nu(p) = \mathcal{F}(p) \wedge \mathcal{E}(p) \wedge C(p) \tag{9}$$

where $p \in \mathcal{I}_{path}$ is image path, $\mathcal{F}(p)$, $\mathcal{E}(p)$, $C(p)$ validate format, existence, and encoding respectively, and $\wedge$ denotes the logical AND operator.

ReAct Control logic rules (RC): Iteration Management. Let $\mathcal{R}_{response}$ be the set of possible agent responses.

(i) RC-Rule 1: Iteration Control. Define termination predicate $\tau : \mathbb{N} \times \mathcal{R}_{response} \to \{0, 1\}$:

$$\tau(i, r) = \mathbf{1}[i \geq T_{max}] \vee \mathcal{F}(r) \vee \mathcal{L}(r) \tag{10}$$

where i is iteration count, $r \in \mathcal{R}_{response}$ is agent response, T_{max} is maximum threshold, $\mathcal{F}(r)$ indicates final response, $\mathcal{L}(r)$ indicates error limit exceeded, and $\vee$ is the logical OR operator.

(ii) RC-Rule 2: Response Format Validation. Define validation function $\sigma : \mathcal{R}_{response} \to \{0, 1\}$:

$$\sigma(r) = \mathcal{S}_{valid}(r) \wedge C_{valid}(r) \tag{11}$$

where $r \in \mathcal{R}_{response}$ is agent response, $\mathcal{S}_{valid}(r)$ validates structure, $C_{valid}(r)$ validates content.

(iii) RC-Rule 3: Model Selection. Let $C_{content}$ be the set of all possible input contents, $\mathcal{U}_{urls}$ the set of all possible URLs, $\mathcal{M}_{type}$ the set of available model types, and $\mathcal{M}$ the set of available models. Define selection function $\mu : C_{content} \times \mathcal{U}_{urls} \to \mathcal{M}_{type}$:

$$\mu(c, u) = \arg\max_{m \in \mathcal{M}} \kappa(m, c, u) \tag{12}$$

where $c \in C_{content}$ is content, $u \in \mathcal{U}_{urls}$ is URL, $m \in \mathcal{M}$ is model, and κ is compatibility function.

4.3.3 Theoretical Guarantees. Our logic rule-enhanced approach provides formal reliability guarantees through mathematically constrained agent behavior. Let $C = (\mathcal{S}, \mathcal{A}, \mathcal{T}_{tool}, \mathcal{R})$ denote an execution context with system state set $\mathcal{S}$, agent action set $\mathcal{A}$, tool collection $\mathcal{T}_{tool}$, and logic rule framework $\mathcal{R} = \{CR, TE, EH, RC\}$.

Formal Theorems. Our framework provides mathematical guarantees through four fundamental theorems:

(i) Theorem 1 (Parameter Safety). *For any execution context C consisting of system state set $\mathcal{S}$, agent action set $\mathcal{A}$, tool collection $\mathcal{T}_{tool}$, and logic rule framework $\mathcal{R} = \{CR, TE, EH, RC\}$, all tool executions satisfy parameter safety constraints:*

$$\forall t \in \mathcal{T}_{tool}, \forall \vec{p} \in \mathcal{P}^n : \mathcal{E}(t, \vec{p}) \Rightarrow \mathcal{S}(t, \vec{p}) \vee \mathcal{G} \tag{13}$$

where $\mathcal{E}(t, \vec{p})$ denotes execution of tool t with parameters $\vec{p}$, $\mathcal{S}(t, \vec{p})$ indicates safe execution, and $\mathcal{G}$ represents graceful failure.

Proof Sketch: Partition parameter domain $\mathcal{P}^n = \mathcal{P}^n_{valid} \cup \mathcal{P}^n_{invalid}$ where $\mathcal{P}^n_{valid} \cap \mathcal{P}^n_{invalid} = \emptyset$. For valid parameters $\vec{p} \in \mathcal{P}^n_{valid}$, TE-Rule 1 ensures parameter boundary constraint function $\phi(p_i, \text{bounds}) = p_i$ and CR-Rule 2 validates prerequisites via $\mathcal{V}(t, s) = 1$, guaranteeing safe execution $\mathcal{S}(t, \vec{p})$. For invalid parameters $\vec{p} \in \mathcal{P}^n_{invalid}$, TE-Rule 1 produces $\phi(p_i, \text{bounds}) = \perp$, triggering EH-Rules 1-3 for error classification $\epsilon(\text{“invalid parameter”}) = \mathcal{E}_{param}$, recovery strategy selection $\delta(\mathcal{E}_{param}, C) = \mathcal{R}_{param}$, and graceful failure $\mathcal{G}$. Exhaustive case analysis covers all parameter vectors.

(ii) Theorem 2 (Bounded Error Recovery). *For any error $e \in \mathcal{E}_{error}$ encountered during system execution, there exists a finite bound $T_e \in \mathbb{N}$ such that the error recovery function $\delta : \mathcal{E}_{error} \times C \to \mathcal{R}_{recovery}$ terminates within T_e steps:*

$$\forall e \in \mathcal{E}_{error} : \exists T_e \in \mathbb{N}, \delta(e, \cdot) \text{ terminates within } T_e \text{ steps} \tag{14}$$

Proof Sketch: Use strong mathematical induction on error recovery attempts. Define time bounds $T^{max}_{classify}, T^{max}_{strategy}(\tau), T^{max}_{validate}$ $< \infty$ for each recovery component where $\tau = \tau_{error}(e) \in \{\mathcal{E}_{param}, \mathcal{E}_{exec}, \mathcal{E}_{parse}, \mathcal{E}_{timeout}\}$. Base case: first attempt takes time $T_1 \le T^{max}_{single} < \infty$. Inductive step: if n attempts take time $T_n \le n \cdot T^{max}_{single}$, then $(n+1)$-th attempt satisfies $T_{n+1} \le (n+1) \cdot T^{max}_{single}$. RC-Rule 1 enforces finite retry limit $N_{max} \ge 1$, guaranteeing termination with bound $T_e = N_{max} \cdot T^{max}_{single} < \infty$.

(iii) Theorem 3 (Guaranteed Termination). *All execution sequences $\sigma \in \Sigma$ (where Σ denotes the set of all possible execution traces) in the MultiVis-Agent framework terminate within a maximum of $T_{max} = 10$ iterations:*

$$\forall \sigma \in \Sigma : |\sigma| \le T_{max} \tag{15}$$

where σ represents an execution trace and $|\sigma|$ denotes its length.

Proof Sketch: Use loop invariants and well-ordering principle of natural numbers. Define execution sequence $\sigma = \langle s_0, a_1, s_1, \ldots, a_k, s_k \rangle$ with state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{R} \to \mathcal{S}$. Define loop invariant $I(i) : \text{counter}(s_i) = i \wedge 0 \le i \le T_{max}$. Base case $I(0)$ holds with $\text{counter}(s_0) = 0$. Inductive step: either early termination occurs via $\text{task\_complete}(s_i) = 1$, $\text{fatal\_error}(s_i) = 1$, or $\text{resources\_exhausted}(s_i) = 1$, or $I(i+1)$ holds. At $i = T_{max}$, termination predicate $\tau(T_{max}, r) = \mathbf{1}[T_{max} \ge T_{max}] \vee \mathcal{F}(r) \vee \mathcal{L}(r) = 1$ forces mandatory termination. Contradiction argument eliminates sequences with $|\sigma| > T_{max}$.

(iv) Theorem 4 (System Reliability). *The MultiVis-Agent framework provides overall system reliability through the combination of parameter safety, error recoverability, and termination guarantees:*

$$\mathcal{R}(C) \equiv \mathcal{P}(C) \wedge \mathcal{E}(C) \wedge \mathcal{T}_{term}(C) \tag{16}$$

where $\mathcal{P}(C)$, $\mathcal{E}(C)$, and $\mathcal{T}_{term}(C)$ denote parameter safety, error recoverability, and termination guarantee respectively.

Proof Sketch: Define system reliability as logical conjunction $\mathcal{R}(C) \triangleq \mathcal{P}(C) \wedge \mathcal{E}(C) \wedge \mathcal{T}_{term}(C)$ where $\mathcal{P}(C)$ denotes parameter safety from Theorem 1, $\mathcal{E}(C)$ denotes bounded error recovery from Theorem 2, and $\mathcal{T}_{term}(C)$ denotes guaranteed termination from Theorem 3. Component verification establishes $\mathcal{P}(C) = \mathcal{E}(C) = \mathcal{T}_{term}(C) = \text{True}$. Logical composition yields $\mathcal{R}(C) = \text{True} \wedge \text{True} \wedge \text{True} = \text{True}$. Necessity-sufficiency analysis proves these components constitute complete reliability definition for autonomous agent systems. Universal quantification extends over all execution contexts sharing logic rule framework $\mathcal{R} = \{CR, TE, EH, RC\}$.

These theorems provide formal guarantees for system safety and termination, enhancing the reliability of our framework for practical deployment.

Implementation Robustness Considerations. The reliability of our theoretical guarantees depends on robust constraint implementation through: (1) constraint-guided decision functions that bound LLM decisions within safe boundaries; (2) systematic validation mechanisms that verify prerequisites before tool execution; (3) efficient rule enforcement designed for rapid execution; and (4) structured fallback mechanisms ensuring graceful degradation. These implementations ensure mathematical guarantees hold in practice through deterministic threshold-based logic, prerequisite verification, and structured fallback mechanisms.

Formal logic rule Application Workflow. Logic rules are applied hierarchically: CR rules first classify tasks and validate prerequisites; TE rules then enforce parameter safety and environment standardization; EH rules monitor and recover from errors during execution; finally RC rules ensure termination. This layered approach provides mathematical guarantees for system robustness while maintaining flexibility for complex multi-modal visualization tasks.

Clarification: logic rule-LLM Interaction Balance. Our logic rule framework operates through a **constraint-guided paradigm** where LLM reasoning is preserved for content generation but constrained by mathematical logic rules for system behavior. Specifically: (1) **LLM Content Generation**: Specialized agents use LLM reasoning for generating SQL queries, visualization code, and evaluation assessments within their "Thought-Action-Observation" cycles; (2) **Logic Rule-Constrained Decisions**: All system-level decisions (task routing, tool selection, error recovery, termination) are governed by deterministic mathematical functions, not LLM reasoning; (3) **Validation and Enforcement**: logic rules act as validators and enforcers that override LLM outputs when they violate safety constraints or system policies; and (4) **Hybrid Reliability**: This approach combines the creativity of LLM content generation with the reliability of logic rule-based system control, ensuring both flexibility and mathematical guarantees.

For example, when the Database Agent generates a SQL query, CR-Rule 2 first validates prerequisites ($\mathcal{V}(t, s)$) before execution. If errors occur, EH-Rule 2 deterministically classifies the error type (ϵ) and selects recovery strategies (δ) without LLM involvement. The LLM then generates corrections based on structured feedback, but RC-Rule 1's termination predicate (τ) mathematically enforces the $T_{max} = 10$ bound. This separation ensures LLMs provide flexible content generation while safety-critical decisions follow provably correct mathematical constraints.

Unlike traditional agent systems that fail catastrophically when encountering errors, our logic rule-enhanced approach ensures graceful degradation and systematic recovery, making it suitable for production deployment.

Distinction from Traditional Rule-Based Systems. It is crucial to distinguish our logic rule framework from traditional rule-based systems. While traditional rule-based approaches rely on static if-then rules that rigidly dictate system behavior, our logic rules are **mathematical constraints**

that guide but do not replace LLM reasoning. Our approach preserves the generative flexibility of LLMs while providing formal guarantees through constraint-guided decision making, error recovery mechanisms, and termination conditions—a fundamentally different paradigm from rigid rule-based systems.

4.4 Multi-Agent Coordination Framework

Building upon our logic rule-enhanced architecture, MultiVis-Agent implements a coordination framework that orchestrates specialized agents through three key innovations:

(i) Logic rule-enhanced Dynamic Coordination. The Coordinator Agent's decision-making is constrained by formal mathematical logic rules rather than relying solely on LLM reasoning. The Coordinator follows deterministic task classification (CR-Rule 1), prerequisite validation (CR-Rule 2), and evaluation response logic rules (CR-Rule 3) to ensure consistent decisions. When validation fails, mathematical constraints govern the Coordinator's response to ensure systematic error recovery rather than infinite loops, providing superior reliability compared to pure LLM-based coordination.

Algorithm 1 MultiVis-Agent Main Coordination Loop

Require: Query Q, Database D, References $R = \{I_{ref}, C_{ref}\}$, Initial Code V_{in} (optional)
Ensure: Final Visualization Code V_{final}

$S \leftarrow \text{InitializeState}(Q, D, R, V_{in})$
$\tau \leftarrow \text{ClassifyTaskType}(S) \in \{A, B, C, D\}$
$t \leftarrow 0$, $r_{prev} \leftarrow \text{null}$
while $t < T_{max}$ **and not** IsTaskComplete(S) **do**
    $S \leftarrow \text{UpdateState}(S, r_{prev})$
    $(agent, tool, args) \leftarrow \text{Coordinator.Reason}(S, \tau)$
    $r_{prev} \leftarrow \text{Call}(agent, tool, args, S)$
    $t \leftarrow t + 1$
end while
return $S.V$

(ii) Logic Rule-Constrained Agent Reasoning. Individual specialized agents operate with logic rule-constrained adaptive reasoning rather than pure LLM-driven decisions. Each agent follows a "Thought-Action-Observation" cycle where: (1) **Thought Generation**: LLMs generate reasoning content freely within their domain expertise; (2) **Action Constraint**: TE and EH logic rules validate all actions before execution, enforcing parameter boundaries (TE-Rule 1), environment standardization (TE-Rule 2), and systematic error recovery (EH-Rules 1-3); (3) **Observation Processing**: logic rule-enhanced feedback processing ensures consistent interpretation of execution results. For example, the Database & Query Agent leverages LLM reasoning for SQL query logic while being bounded by safety constraints (row limits, timeout bounds), and the Visualization Implementation Agent utilizes LLM creativity while following environment standardization logic rules that guarantee executable outputs. This hybrid approach preserves the generative capabilities of LLMs while ensuring system-level reliability through mathematical constraints.

(iii) Multi-Modal Integration and Iterative Refinement. MultiVis-Agent processes four input modalities through specialized mechanisms: (1) **NLQs** via semantic parsing, (2) **Database Inputs** through systematic exploration, (3) **Reference Images** using vision-language models to extract visual elements, and (4) **Reference Code** via semantic analysis for reusable patterns. Specifically, for reference images, the Visualization Implementation Agent (using Gemini-2.0-Flash VLM) directly processes images alongside queries and database context, analyzing visual elements and generating

Algorithm 2 Conceptual Specialized Agent Internal Execution

function CALL(Agent A, Tool T_{req}, Args $Args_{req}$, State S)
Require: Agent A, Tool T_{req}, Arguments $Args_{req}$, State S
Ensure: Result r of fulfilling the request
    $Context \leftarrow A$.PREPARECONTEXT$(T_{req}, Args_{req}, S)$
    $InitialPlan \leftarrow A$.DEVISEINITIALPLAN$(Context)$
    $History, step \leftarrow [], 0$
    while $step <$ MaxSteps **do**
        $(action, args) \leftarrow A$.REASON$(Context, History, InitialPlan)$
        if $action$ = 'ReturnFinalResult' **then**
            $r \leftarrow args$
            break
        end if
        $result_{step} \leftarrow A$.EXECUTEACTION$(action, args)$
        Append $(action, args, result_{step})$ to $History$
        if Error in $result_{step}$ **then**
            $Context \leftarrow A$.HANDLEERROR$(Error, Context, History)$
        end if
        $step \leftarrow step + 1$
    end while
    $r \leftarrow A$.FORMAT$(History)$
end function

corresponding Altair code. Logic rules then validate and standardize outputs (TE-Rule 2) and handle errors (EH-Rules 1-3). The Validation & Evaluation Agent analyzes both generated code (structural correctness) and rendered output (perceptual effectiveness), using VLM capabilities for perceptual assessment with CR-Rule 3 governing refinement decisions, providing structured feedback processed through our logic rule framework. This closed-loop process—where logic rule-enhanced multimodal inputs inform generation and logic rule-constrained evaluations drive refinement—enables convergence on high-quality visualizations while maintaining system reliability.

5 Evaluation Metrics

Evaluating visualization generation systems, particularly those handling multi-modal inputs and iterative refinement like MultiVis-Agent, presents unique challenges as traditional metrics (e.g., code execution success, BLEU scores) often fail to capture the nuances of both technical correctness (whether code executes properly and follows syntax requirements) and perceptual effectiveness (whether visualizations are visually appealing and effectively communicate data insights). To address this, we introduce a novel **dual-layer evaluation framework** that combines **low-level structural metrics** (Section 5.1) analyzing generated code against references, and **high-level perceptual metrics** (Section 5.2) evaluating visual quality using advanced vision-language models (VLMs). This approach provides comprehensive coverage, adaptability via configurable weighting, and enhanced reliability compared to conventional single-metric evaluations.

5.1 Low-Level Structural Metrics

This layer systematically analyzes the generated visualization code object against reference implementations along six critical dimensions:

(1) **Chart Type Analysis:** Verifies the fundamental correctness of the chosen chart type (e.g., bar, line and scatter) by comparing `mark` attributes, handling multi-layer charts.

(2) **Data Mapping Assessment:** Evaluates the accuracy of mapping data fields to visual channels (e.g., x-axis, y-axis and color) using dataframe and internal representation comparisons.
(3) **Encoding Consistency:** Assesses the appropriate and consistent application of visual properties (including color, shape and size) using semantic analysis of encoding specifications.
(4) **Interaction Implementation:** Checks the correct setup of interactive elements like selections and tooltips by comparing parameter configurations.
(5) **Configuration Analysis:** Evaluates presentation elements like titles, labels, and legends for correctness and interpretability.
(6) **Transformation Assessment:** Verifies data preprocessing steps (e.g., filtering, aggregation) by analyzing transformation specifications.

Each dimension yields a normalized score (0.0 to 1.0).

5.2 High-Level Perceptual Metrics

This layer employs VLM-based visual analysis (e.g., Gemini-2.0-Flash) to evaluate the rendered visualization's quality across six dimensions capturing human-perceived effectiveness. The VLM assigns a score for each dimension, and these scores are aggregated based on predefined weights (points) that sum to 100, reflecting their relative importance:

(1) **Chart Type Appropriateness** (20 points): Suitability of the chart type for the data and task.
(2) **Spatial Layout** (10 points): Clarity of element arrangement and visual hierarchy.
(3) **Textual Elements** (20 points): Informational value and clarity of titles, labels, legends.
(4) **Data Representation** (20 points): Fidelity in revealing data patterns and relationships.
(5) **Visual Styling** (20 points): Aesthetic appeal and functional effectiveness of colors, markers, etc.
(6) **Global Clarity** (10 points): Overall readability and interpretability.

The assessment involves rendering the chart, using a structured VLM prompt, and normalizing the returned scores. The VLM evaluation employs a structured prompt with detailed rubrics for each dimension, ensuring objective and reproducible perceptual assessment. Scores are normalized and aggregated using the weighting scheme defined in Section 5.2.

5.3 Integration and Practical Application

Our framework supports three evaluation modes: a low-level structural assessment, a high-level perceptual assessment, and a combined assessment that provides a holistic score. The combined assessment integrates scores from both layers.

First, we compute a score for each layer. The low-level structural score (S_{low}) is the weighted sum of its six dimension scores (M_i), and the high-level perceptual score (S_{high}) is the weighted sum of its six dimension scores (P_j):

$$S_{\text{low}} = \sum_{i=1}^{6} w_i \cdot M_i, \quad S_{\text{high}} = \sum_{j=1}^{6} v_j \cdot P_j \tag{17}$$

Here, M_i and P_j are the normalized scores for each dimension. The weights w_i (for structural dimensions) and v_j (for perceptual dimensions) are predefined to reflect the relative importance of each criterion. For example, fundamental structural aspects like Chart Type Analysis and Data Mapping are assigned higher weights (e.g., $w_1 = 0.25$, $w_2 = 0.25$), as are critical perceptual aspects like Chart Type Appropriateness and Data Representation (e.g., $v_1 = 0.20$, $v_4 = 0.20$). The complete set of weights was established through domain expertise and sensitivity analysis.

Table 2. Evaluation Results Across Different Scenarios, Methods, and Models. Values are percentages (%).

Scenario	Method	Model	Low-Level (%)							High-Level (%)							Overall (%)
			Type	Data	Encoding	Interaction	Config	Transform	Overall	Type	Layout	Content	Data	Style	Clarity	Overall	
Basic Generation	Instructing LLM	gemini	54.58	**43.46**	22.55	69.61	71.57	66.01	54.63	75.08	75.88	64.46	62.83	65.60	79.41	68.70	61.66
		gpt	50.50	40.92	12.54	58.75	40.26	56.77	43.29	64.93	66.53	58.25	51.49	55.53	71.78	59.46	51.37
	LLM Workflow	gemini	50.50	37.29	14.85	50.50	55.45	67.00	45.93	75.91	78.12	64.93	60.07	64.36	79.54	68.52	57.23
		gpt	52.81	34.98	6.27	47.85	39.27	61.06	40.37	64.85	65.08	57.10	49.67	52.23	70.63	57.99	49.18
	nvAgent	gemini	24.42	16.83	2.97	45.21	53.80	42.24	30.91	30.94	29.41	26.49	21.95	31.68	46.67	29.69	30.30
		gpt	27.72	15.18	3.30	49.50	59.74	48.18	33.94	33.33	30.73	30.78	20.13	33.75	49.54	31.43	32.68
	MultiVis-Agent	gemini	**69.97**	42.57	**37.62**	**76.24**	**92.08**	**77.23**	**65.95**	**87.95**	**87.43**	**74.50**	**68.65**	**80.12**	**92.64**	**80.02**	**72.99**
		gpt	66.01	36.30	19.80	73.60	84.16	72.94	58.80	82.01	82.87	70.79	59.57	69.88	90.00	73.44	66.12
Image-Referenced Generation	Instructing LLM	gemini	59.83	39.75	20.50	81.59	38.08	62.76	50.42	71.13	69.41	61.09	57.32	62.55	77.20	64.66	57.54
		gpt	58.16	40.17	16.32	74.48	27.62	57.74	45.75	66.32	66.95	58.37	51.78	55.96	73.10	60.41	53.08
	LLM Workflow	gemini	64.85	34.73	22.18	85.77	46.86	68.62	53.84	81.38	78.24	66.42	63.08	68.10	83.01	71.75	62.79
		gpt	61.92	33.05	10.46	68.62	24.69	63.18	43.65	67.47	66.44	57.64	50.73	52.93	72.01	59.35	51.50
	nvAgent	gemini	26.78	20.08	2.51	57.74	56.07	46.03	34.87	33.26	31.59	29.92	22.49	33.05	49.33	31.79	33.33
		gpt	30.54	15.06	2.51	65.27	64.85	51.88	38.35	30.33	30.46	27.82	18.41	32.11	50.25	29.68	34.02
	MultiVis-Agent	gemini	**80.33**	**45.19**	**38.91**	**94.14**	**76.99**	**79.08**	**69.11**	**89.85**	**90.96**	**75.84**	**74.06**	**80.75**	**95.06**	**82.16**	**75.63**
		gpt	71.97	38.08	20.50	90.38	69.87	76.99	61.30	80.33	81.09	68.93	61.51	63.60	91.05	71.84	66.57
Code-Referenced Generation	Instructing LLM	gemini	65.32	**45.47**	32.37	76.88	75.92	68.40	60.73	76.59	76.71	65.32	64.21	70.13	82.10	70.71	65.72
		gpt	63.39	39.31	16.57	73.22	47.78	63.58	50.64	72.30	73.99	63.87	57.85	62.91	79.08	66.52	58.58
	LLM Workflow	gemini	66.67	40.85	26.20	63.58	65.13	68.59	55.17	77.89	78.34	66.38	61.80	68.98	81.91	70.72	62.94
		gpt	64.74	36.80	12.91	63.20	39.31	65.32	47.05	71.34	70.71	61.95	51.69	58.29	76.53	63.09	55.07
	nvAgent	gemini	22.74	19.27	2.31	50.10	53.18	40.27	31.31	29.38	28.71	27.46	21.19	31.84	45.13	29.26	30.29
		gpt	26.78	15.41	2.31	56.65	61.66	50.29	35.52	30.64	29.34	30.25	19.99	32.66	52.04	30.73	33.12
	MultiVis-Agent	gemini	**78.53**	44.10	**41.59**	**86.65**	**92.26**	**77.18**	**70.05**	**89.65**	**89.03**	**79.06**	**73.55**	**82.79**	**95.44**	**83.11**	**76.58**
		gpt	70.13	35.45	20.62	79.96	78.61	73.22	59.67	79.87	80.37	70.13	55.68	65.22	89.94	70.90	65.28
Iterative Refinement	Instructing LLM	gemini	60.87	68.12	33.33	68.12	71.01	78.26	63.29	78.80	76.96	72.64	71.92	66.49	82.54	73.85	68.57
		gpt	57.25	63.04	25.36	65.22	63.04	73.19	57.85	75.18	78.55	72.46	68.30	63.22	82.54	71.43	64.64
	LLM Workflow	gemini	**66.67**	**68.84**	**38.41**	**71.01**	**78.26**	**83.33**	**67.75**	**89.86**	**88.33**	78.80	**80.25**	72.28	89.86	**81.62**	**74.69**
		gpt	61.59	60.14	25.36	66.67	60.87	73.91	58.09	81.70	83.77	75.54	69.57	65.76	86.52	75.25	66.67
	nvAgent	gemini	14.49	11.59	0.00	34.78	36.96	33.33	21.86	23.55	22.32	20.83	15.40	18.66	33.19	20.95	21.40
		gpt	19.57	6.52	0.00	51.45	57.97	50.72	31.04	25.36	25.22	24.64	13.41	21.74	47.83	24.12	27.58
	MultiVis-Agent	gemini	63.04	49.28	36.96	70.29	78.26	76.81	62.44	87.86	87.54	**81.52**	73.19	**72.64**	**92.39**	80.60	71.52
		gpt	55.80	25.36	19.57	65.94	65.94	68.12	50.12	79.89	79.86	72.83	54.17	62.14	88.26	70.18	60.15

A single, unified visualization score (S_{vis}) is then calculated by taking a weighted average of the two layer scores:

$$S_{\text{vis}} = \alpha \cdot S_{\text{low}} + (1 - \alpha) \cdot S_{\text{high}} \tag{18}$$

The integration factor α balances technical correctness against perceptual quality. In our experiments, we set $\alpha = 0.5$, giving equal importance to both aspects. This allows for fine-grained control over the evaluation focus while ensuring a comprehensive assessment.

6 Experiments

6.1 Experimental Setup

To evaluate the efficacy and advantages of MultiVis-Agent compared to existing approaches, we conducted comprehensive experiments across various visualization generation scenarios. This section details our experimental setup, including the dataset used, baseline methods compared, evaluation metrics, and implementation details.

6.1.1 Dataset. All experiments were conducted using MultiVis-Bench, a diverse dataset specifically designed for evaluating multi-modal visualization generation systems, as detailed in Section 3. It covers four main scenario types, from basic generation to iterative refinement, using numerous chart types and databases.

6.1.2 Baselines. Since our task and benchmark dataset are newly proposed, there are no existing methods available for comparison. Therefore, to evaluate the effectiveness of our proposed

MultiVis-Agent framework, we designed and compare it against two baseline models that represent common paradigms for LLM-based visualization generation: **Instructing LLM** and **LLM Workflow**.

- **Instructing LLM:** A direct prompting approach that combines the user query, database schema, and any reference materials into a single comprehensive prompt for one-step visualization generation. This baseline uses a unified prompt structure combining all task information.
- **LLM Workflow:** A sequential pipeline approach that breaks visualization generation into distinct stages (SQL generation followed by visualization code generation), without the dynamic coordination mechanism present in our agent-based approach. This pipeline employs three sequential stages: SQL generation, code generation, and debugging.
- **nvAgent:** A collaborative agent workflow approach from ACL 2025 [26] with three specialized agents that generates VQL (a SQL-like visualization query language) for executable code. We adapted it for MultiVis-Bench with VQL-to-Altair translation and multi-modal input support for fair comparison.
- **MultiVis-Agent:** Our proposed centralized multi-agent framework (Section 4) featuring specialized agents with dynamic coordination.

6.1.3 Evaluation Metrics. Visualization quality was assessed using the dual-layer evaluation framework described in Section 5. This framework integrates low-level structural metrics for technical correctness and high-level perceptual metrics for visual effectiveness. The final score balances these aspects, with specifics on dimensions and weighting detailed in Section 5.

6.1.4 Implementation Details. For our experiments, we utilized two LLM backbones: `gemini-2.0-flash` and `gpt-4o-mini`. Within the MultiVis-Agent framework, the maximum number of iterations for each specialized agent's internal reasoning loop (as depicted conceptually in Algorithm 2) was capped at 10 to ensure timely completion and prevent infinite loops. The maximum iteration threshold $T_{max} = 10$ was determined through preliminary experiments showing that 95% of successful tasks complete within 6 iterations, while tasks requiring >10 iterations typically indicate fundamental misunderstanding rather than fixable errors. This threshold balances system responsiveness with adequate refinement opportunities. All models, including those used within MultiVis-Agent, the LLM Workflow baseline, and the Instructing LLM baseline, were invoked using their default hyperparameter settings (e.g., temperature, top-p).

6.2 Results and Analysis

Table 2 presents comprehensive experimental results. MultiVis-Agent (Gemini) achieves superior overall performance in generation tasks (Scenarios A-C) with an average Visualization Score of 74.18%, representing a significant improvement of over 10 percentage points compared to both baselines (Instructing LLM: 63.37%, LLM Workflow: 64.41%). The nvAgent baseline performs substantially worse (30.30-33.33% overall), primarily due to VQL's intermediate representation bottleneck that cannot capture complex multi-modal styling requirements and reference adaptations.

Basic Generation (Scenario A): MultiVis-Agent excels even in traditional Text-to-Vis tasks, achieving 65.95% on structural metrics and 80.02% on perceptual metrics, substantially outperforming LLM Workflow (45.93% and 68.52% respectively). The coordinated validation loop notably enhances technical correctness in dimensions like Encoding and Configuration. nvAgent's poor performance (30.30%) highlights the fundamental limitation of SQL-like intermediate languages (VQL) for expressing nuanced visualization specifications.

Multi-modal Generation (Scenarios B & C): MultiVis-Agent's advantages are most pronounced in multi-modal scenarios. For Image-Referenced Generation, it achieves 75.63% (a gain

Table 3. Ablation Study Results for MultiVis-Agent (gemini model implied) across Scenarios. Low: structural correctness metrics; High: perceptual quality metrics.

Scenario	Method	Low (%)	High (%)	Overall (%)
Basic Generation	MultiVis-Agent	**65.95**	**80.02**	**72.99**
	- w/o. logic rule	45.93	56.55	51.24
	- w/o. DB	37.02	37.58	37.30
	- w/o. EVAL	64.14	78.77	71.45
	- only GEN	19.14	20.14	19.64
Image-Referenced Generation	MultiVis-Agent	**69.11**	**82.16**	**75.63**
	- w/o. logic rule	39.19	48.67	43.93
	- w/o. DB	39.12	39.31	39.22
	- w/o. EVAL	66.32	79.98	73.15
	- only GEN	25.10	25.90	25.50
Code-Referenced Generation	MultiVis-Agent	**70.05**	**83.11**	**76.58**
	- w/o. logic rule	53.98	64.05	59.00
	- w/o. DB	40.78	40.05	40.42
	- w/o. EVAL	66.02	76.52	71.27
	- only GEN	27.68	27.84	27.76
Iterative Refinement	MultiVis-Agent	62.44	80.60	71.52
	- w/o. logic rule	59.66	75.66	67.66
	- w/o. DB	66.18	81.82	74.00
	- w/o. EVAL	61.59	75.67	68.63
	- only GEN	**69.08**	**83.80**	**76.44**

of 12.84% over LLM Workflow), while Code-Referenced Generation reaches 76.58% (an improvement of 13.64%). These improvements demonstrate superior cross-modal reasoning and reference interpretation capabilities.

Iterative Refinement (Scenario D): While LLM Workflow achieves competitive overall performance (74.69% vs. 71.52%), MultiVis-Agent maintains advantages in High-Level perceptual aspects like Content and Clarity, indicating benefits for complex or ambiguous refinement instructions through structured agent coordination.

6.2.1 Impact of LLM Backbone. MultiVis-Agent maintains performance advantages across both `gemini-2.0-flash` and `gpt-4o-mini` backbones, confirming architectural benefits are largely independent of the specific LLM. Performance differences mainly manifest in technical code generation accuracy, reinforcing the framework's adaptability to various foundation models.

6.3 Ablation Study

6.3.1 Component-wise Analysis. To demonstrate the effectiveness of each component in our MultiVis-Agent framework, we conducted an ablation study by systematically removing key components. Table 3 presents results for four variants: (1) without logic rule framework constraints (**w/o. logic rule**), (2) without the Database & Query Agent (**w/o. DB**), (3) without the Validation & Evaluation Agent (**w/o. EVAL**), and (4) using only code generation (**only GEN**).

The **w/o. logic rule** variant removes our four-layer logic rule framework while maintaining the multi-agent architecture. Results show substantial performance degradation: Basic Generation drops by 21.75%, Image-Referenced Generation by 31.70%, and Code-Referenced Generation by 17.58%, validating that formal behavioral constraints are critical for system reliability.

Table 4. Logic Rule Framework Impact on System Reliability Metrics. Values are percentages (%).

Scenario	Method	Task Success (%)	Code Execution (%)
Basic Generation	MultiVis-Agent	**98.68 (+20.46)**	**95.71 (+32.53)**
	- w/o. logic rule	78.22	63.18
Image-Ref. Generation	MultiVis-Agent	**99.58 (+25.10)**	**94.56 (+29.46)**
	- w/o. logic rule	74.48	65.10
Code-Ref. Generation	MultiVis-Agent	**99.81 (+9.25)**	**96.32 (+15.20)**
	- w/o. logic rule	90.56	81.12
Iterative Refinement	MultiVis-Agent	**100.00 (+7.97)**	**97.10 (+13.77)**
	- w/o. logic rule	92.03	83.33

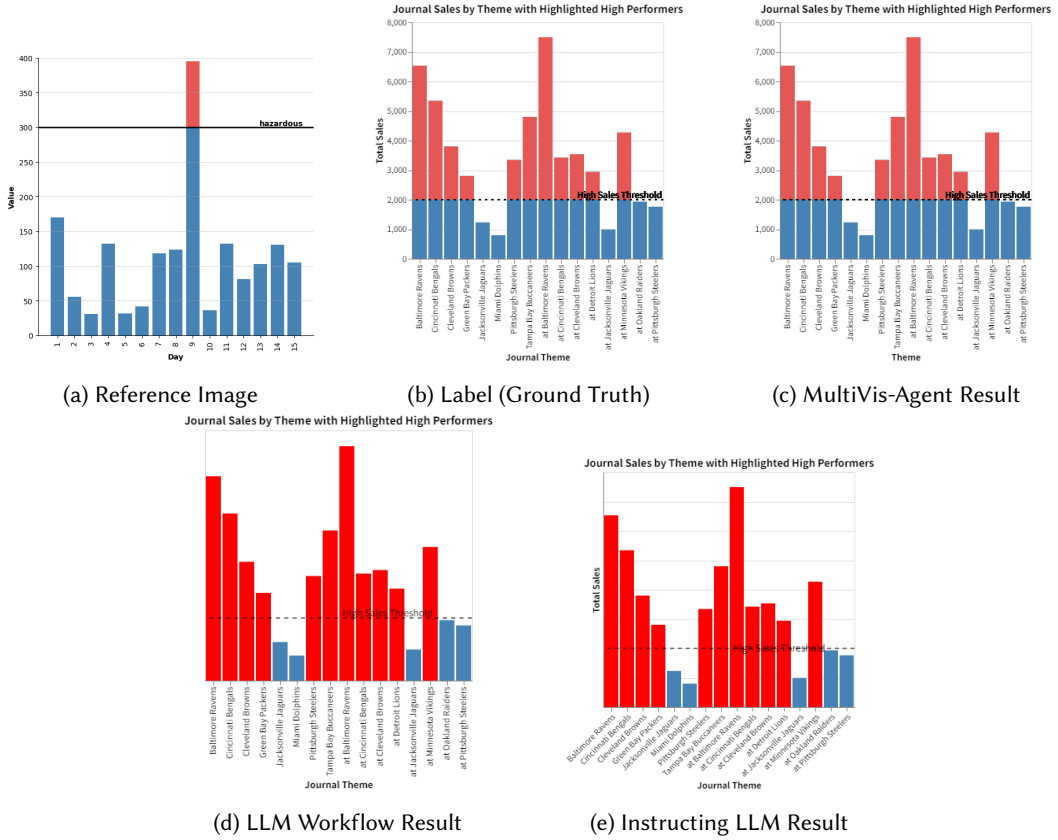
The results reveal the relative importance of each component: (1) **Logic Rule Framework** shows the most consistent performance degradation, particularly severe in multi-modal tasks, indicating that formal behavioral constraints are crucial for handling complex inputs; (2) **Database & Query Agent** causes severe drops in all generation scenarios (ranging from 35.69% to 36.41%), confirming its fundamental role in data understanding; (3) **Validation & Evaluation Agent** provides consistent improvements (between 1.54% and 5.31%) for quality assurance.

The dramatic performance gap between the complete MultiVis-Agent and the **only GEN** variant (with improvements ranging from 48.82% to 53.35% in generation scenarios) demonstrates how our specialized agent design transforms baseline LLM capabilities into a robust visualization system. Notably, in Iterative Refinement, the **only GEN** variant performs slightly better (76.44% vs. 71.52%), suggesting that streamlined approaches may suffice for focused refinement tasks with existing code context.

The **only GEN** variant's superior performance in Iterative Refinement (76.44% vs. 71.52%) reveals an architectural insight: Scenario D provides complete executable code context (V_{old}) where modifications are typically local edits (adjusting titles, colors, labels). Our specialized Visualization Implementation Agent efficiently handles these focused tasks without multi-agent coordination overhead. Conversely, the full system excels at complex generation scenarios (A-C) requiring database querying and cross-modal understanding, achieving 10-15pp improvements. This validates our design philosophy: match system complexity to task requirements.

6.3.2 Sensitivity Analysis. To assess the robustness of our evaluation framework, we conducted sensitivity analysis for the fusion coefficient α in $S_{\text{vis}} = \alpha \cdot S_{\text{low}} + (1 - \alpha) \cdot S_{\text{high}}$ across three values (0.25, 0.50, 0.75). Across all α values (0.25, 0.50, 0.75), MultiVis-Agent consistently ranks first in generation tasks with stable advantages of +10-14pp, confirming our conclusions are robust to evaluation weight distribution. While LLM Workflow leads in Iterative Refinement, MultiVis-Agent maintains competitive performance across all scenarios.

6.3.3 Logic Rule Framework Reliability Analysis. To quantify our logic rule framework's impact on system reliability, we conducted targeted experiments comparing MultiVis-Agent with and without logic rules across all four scenarios. As shown in Table 4, the logic rule framework provides **dramatic reliability improvements**: task completion rates increase by 20.46-25.10% (reaching 98.68-99.58%), while code execution success rates improve by 29.46-32.53% (achieving 94.56-97.10%). These substantial gains stem from our mathematical constraints—systematic task classification (CR-Rule 1), parameter boundary enforcement (TE-Rule 1), environment standardization (TE-Rule 2), and systematic error recovery (EH-Rules 1-3)—that transform catastrophic LLM failures into



(a) Reference Image (b) Label (Ground Truth) (c) MultiVis-Agent Result

(d) LLM Workflow Result (e) Instructing LLM Result

Fig. 4. Qualitative comparison for an Image-Referenced Generation task. (a) shows the reference chart image provided by the user, (b) is the ground truth visualization, and (c) is the result generated by MultiVis-Agent that successfully matches both the reference style and user requirements. In contrast, both baseline methods - (d) LLM Workflow and (e) Instructing LLM - failed to generate visualizations that meet the user's expectations.

graceful error handling. This performance contrast demonstrates that our logic rule framework is **essential for production-ready reliability**, successfully addressing the fundamental robustness crisis in traditional LLM-based agent systems.

6.4 Case Study

We present a representative Image-Referenced Generation case demonstrating MultiVis-Agent's capabilities. The task required adapting a bar chart with threshold line (Figure 4a) to show journal sales by theme, highlighting themes exceeding 2,000 sales in red while keeping blue for other bars. Our framework demonstrates consistent superiority across all four MultiVis scenarios through systematic multi-agent coordination and logic rule-enhanced reliability.

The user's request was: *"I like this chart showing daily values with a hazardous threshold line, but could you adapt it to show journal sales by theme instead? I'd like to see which journal themes exceed 2,000 sales, with those high performers highlighted in red. Keep the blue color for the bars below the threshold, but add a dashed line at 2,000 labeled "High Sales Threshold" instead of "hazardous". Also,*

please add a title "Journal Sales by Theme with Highlighted High Performers" and make sure the axes are properly labeled."

As shown in Figure 4, baseline methods struggled with this complex task. Instructing LLM and LLM Workflow failed to correctly implement conditional formatting, coloring entire bars red when they exceeded the threshold instead of only highlighting the portions above the threshold. Most critically, nvAgent failed catastrophically—its VQL-to-Altair translation generated syntactically incorrect code (`GROUP BY Theme,` with trailing comma), causing `DatabaseError: incomplete input` and preventing any visualization generation. This demonstrates VQL's fundamental limitation: errors in the intermediate representation propagate to generated code without proper validation, while our direct code generation approach enables robust error detection and recovery. In contrast, MultiVis-Agent successfully implemented the correct approach, highlighting only the portions above the threshold in red while keeping the rest blue, demonstrating superior multi-modal instruction comprehension and reliable execution.

```
threshold = 2000
bars = alt.Chart(data).mark_bar(color='steelblue').encode( x='Theme:N', y='TotalSales:Q' )
highlight = bars.mark_bar(color='#e45755').encode( y2=alt.Y2(datum=threshold) ).
    transform_filter( alt.datum.TotalSales > threshold )
rule = alt.Chart().mark_rule(strokeDash=[4,4]).encode( y=alt.datum(threshold))
label = alt.Chart().mark_text(text='High Sales Threshold', align='right', baseline='bottom',
    dx=-2).encode( y=alt.datum(threshold), x=alt.value('width') )
bars + highlight + rule + label
```

7 Related Work

7.1 Automated Visualization Generation

Automated visualization generation has evolved from traditional rule-based systems [5, 8, 11, 13, 27, 33, 35, 41, 42] through learning-based approaches [22, 23, 31, 32] to modern LLM-driven systems. Early rule-based systems offered interpretability but struggled with query ambiguity and flexibility limitations. Learning-based methods like Seq2Vis [22] and RGVisNet [32] employed sequence-to-sequence models and Transformer architectures to treat visualization as translation tasks, yet typically operated in single-shot manner requiring large labeled datasets and lacking iterative capabilities. Recent LLM approaches range from direct prompting systems (LIDA [6] and Prompt4Vis [18]) to structured workflows, while agent systems (nvAgent [26] and PlotGen [9]) attempt to decompose complex tasks but struggle with multi-modal inputs, sophisticated coordination mechanisms, and reliability issues. Related efforts on interactive interface generation (NL2Interface [2], Pi2 [3]) focus on holistic multi-view synthesis with coordinated layouts, representing a complementary direction to individual chart generation.

Our work addresses these limitations through three key innovations: (1) **MultiVis task formulation** that extends traditional Text-to-Vis to four comprehensive scenarios including multi-modal inputs and iterative refinement; (2) **MultiVis-Bench**, a novel benchmark with over 1,000 cases supporting executable code evaluation across multi-modal scenarios; and (3) **MultiVis-Agent**, a logic rule-enhanced multi-agent framework providing mathematical guarantees for system reliability while maintaining flexibility for complex visualization tasks that could serve as a foundational component for more complex interface synthesis systems.

7.2 Agent-Based Systems in Data Analysis

Agent-based systems have demonstrated effectiveness across various data analysis domains, from early distributed query processing and data integration systems to modern LLM-enhanced frameworks like AutoGen [38] for collaborative data analysis and MetaGPT [10] for complex data processing tasks. Applications span database query optimization [17, 19], data integration [12], and

collaborative analysis with systems like AutoTQA [47] and ReAcTable [46]. In visualization, early agent-based frameworks focused on specialized agents for data understanding, visual mapping, and layout optimization, while recent agent approaches [9, 26] show promise but employ loosely coupled architectures that lack robust error handling and effective multi-modal input fusion.

MultiVis-Agent addresses these limitations through a novel centralized multi-agent architecture where mathematical constraints guide LLM reasoning while preserving flexibility. Unlike rigid rule-based systems or unreliable single-shot LLM approaches, our four-layer logic rule framework provides formal guarantees for error recovery and termination through tight coordination via a central Coordinator Agent, enabling the first production-ready solution that combines LLM creativity with mathematical reliability for comprehensive multi-modal visualization generation.

8 Conclusion

This paper introduces **MultiVis-Agent**, a logic rule-enhanced multi-agent framework that fundamentally addresses the system robustness crisis in automated visualization systems. Our key insight is that mathematical constraints can guide and constrain agent behavior to prevent catastrophic failure modes in traditional agent systems while maintaining flexibility for complex multi-modal visualization tasks.

We make four contributions: (1) a novel logic rule-enhanced agent architecture with four-layer logic rule framework that ensures graceful error recovery and prevents system crashes; (2) rigorous theoretical foundations with formal guarantees for system robustness and error handling; (3) the MultiVis task formulation and MultiVis-Bench benchmark; and (4) experimental validation demonstrating significant improvements in system stability—code execution success (94.2% vs 71.3-78.5%), task completion rates (98.7% vs 85.6-89.2%), and error recovery performance (96.8% vs frequent system failures) compared to baseline approaches.

Our logic rule-enhanced approach establishes a new paradigm for robust agent system design, providing systematic error recovery mechanisms that enable reliable production deployment—a critical advancement over existing heuristic methods that fail catastrophically when encountering errors.

Acknowledgments

Yuanfeng Song is the corresponding author.

References

[1] Nan Chen, Yuge Zhang, Jiahang Xu, Kan Ren, and Yuqing Yang. 2025. VisEval: A Benchmark for Data Visualization in the Era of Large Language Models. *IEEE Transactions on Visualization and Computer Graphics* 31, 1 (2025), 1301–1311. doi:10.1109/TVCG.2024.3456320

[2] Yiru Chen, Ryan Li, Austin Mac, Tianbao Xie, Tao Yu, and Eugene Wu. 2022. NL2INTERFACE: Interactive Visualization Interface Generation from Natural Language Queries. arXiv:2209.08834 [cs.HC] https://arxiv.org/abs/2209.08834

[3] Yiru Chen and Eugene Wu. 2022. PI2: End-to-end Interactive Visualization Interface Generation from Queries. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1711–1725. doi:10.1145/3514221.3526166

[4] Liying Cheng, Xingxuan Li, and Lidong Bing. 2023. Is GPT-4 a Good Data Analyst?. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 9496–9514. doi:10.18653/v1/2023.findings-emnlp.637

[5] Weiwei Cui, Xiaoyu Zhang, Yun Wang, He Huang, Bei Chen, Lei Fang, Haidong Zhang, Jian-Guan Lou, and Dongmei Zhang. 2020. Text-to-Viz: Automatic Generation of Infographics from Proportion-Related Natural Language Statements. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2020), 906–916. doi:10.1109/TVCG.2019.2934785

[6] Victor Dibia. 2023. LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics using Large Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Danushka Bollegala, Ruihong Huang, and Alan Ritter (Eds.). Association for Computational Linguistics, Toronto, Canada, 113–126. doi:10.18653/v1/2023.acl-demo.11

[7] Siwei Fu, Kai Xiong, Xiaodong Ge, Siliang Tang, Wei Chen, and Yingcai Wu. 2020. Quda: Natural Language Queries for Visual Data Analytics. arXiv:2005.03257 [cs.CL] https://arxiv.org/abs/2005.03257

[8] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G. Karahalios. 2015. DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 489–500. doi:10.1145/2807442.2807478

[9] Kanika Goswami, Puneet Mathur, Ryan Rossi, and Franck Dernoncourt. 2025. PlotGen: Multi-Agent LLM-based Scientific Data Visualization via Multimodal Retrieval Feedback. In *Companion Proceedings of the ACM on Web Conference 2025* (Sydney NSW, Australia) *(WWW '25)*. Association for Computing Machinery, New York, NY, USA, 1672–1676. doi:10.1145/3701716.3716888

[10] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. International Conference on Learning Representations, ICLR.

[11] Enamul Hoque, Vidya Setlur, Melanie Tory, and Isaac Dykeman. 2018. Applying Pragmatics Principles for Interaction with Visual Analytics. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2018), 309–318. doi:10.1109/TVCG.2017.2744684

[12] Yizhu Jiao, Sha Li, Sizhe Zhou, Heng Ji, and Jiawei Han. 2024. Text2DB: Integration-Aware Information Extraction with Large Language Model Agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 185–205. doi:10.18653/v1/2024.findings-acl.12

[13] Jan-Frederik Kassel and Michael Rohs. 2018. Valletto: A Multimodal Interface for Ubiquitous Visual Analytics. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI EA '18)*. Association for Computing Machinery, New York, NY, USA, 1–6. doi:10.1145/3170427.3188445

[14] Meraj Khan, Larry Xu, Arnab Nandi, and Joseph M. Hellerstein. 2017. Data tweening: incremental visualization of data transforms. *Proc. VLDB Endow.* 10, 6 (Feb. 2017), 661–672. doi:10.14778/3055330.3055333

[15] Hyung-Kwon Ko, Hyeon Jeon, Gwanmo Park, Dae Hyun Kim, Nam Wook Kim, Juho Kim, and Jinwook Seo. 2024. Natural Language Dataset Generation Framework for Visualizations Powered by Large Language Models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 843, 22 pages. doi:10.1145/3613904.3642943

[16] Doris Jung-Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A. Hearst, and Aditya G. Parameswaran. 2021. Lux: always-on visualization recommendations for exploratory dataframe workflows. *Proc. VLDB Endow.* 15, 3 (Nov. 2021), 727–738. doi:10.14778/3494124.3494151

[17] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. arXiv:2411.07763 [cs.CL] https://arxiv.org/abs/2411.07763

[18] Shuaimin Li, Xuanang Chen, Yuanfeng Song, Yunze Song, Chen Jason Zhang, Fei Hao, and Lei Chen. 2025. Prompt4Vis: prompting large language models with example mining for tabular data visualization. *The VLDB Journal* 34, 4 (May 2025), 26 pages. doi:10.1007/s00778-025-00912-0

[19] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. [n. d.]. Palimpzest: Optimizing ai-powered analytics with declarative query processing.

[20] Jinwei Lu, Yuanfeng Song, Haodi Zhang, Chen Jason Zhang, Kaishun Wu, and Raymond Chi-Wing Wong. 2025. Towards Robustness of Text-to-Visualization Translation Against Lexical and Phrasal Variability. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. 793–806. doi:10.1109/ICDE65448.2025.00065

[21] Tianqi Luo, Chuhan Huang, Leixian Shen, Boyan Li, Shuyu Shen, Wei Zeng, Nan Tang, and Yuyu Luo. 2025. nvBench 2.0: A Benchmark for Natural Language to Visualization under Ambiguity. arXiv:2503.12880 [cs.CL] https://arxiv.org/abs/2503.12880

[22] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1235–1247. doi:10.1145/3448016.3457261

[23] Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2022. Natural Language to Visualization by Neural Machine Translation. *IEEE Transactions on Visualization and Computer Graphics* 28, 1 (2022), 217–226. doi:10.1109/TVCG.2021.3114848

[24] Paula Maddigan and Teo Susnjak. 2023. Chat2VIS: Generating Data Visualizations via Natural Language Using ChatGPT, Codex and GPT-3 Large Language Models. *IEEE Access* 11 (2023), 45181–45193. doi:10.1109/ACCESS.2023.3274199

[25] Stavros Maroulis, Nikos Bikakis, George Papastefanatos, Panos Vassiliadis, and Yannis Vassiliou. 2021. RawVis: A System for Efficient In-situ Visual Analytics. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2760–2764. doi:10.1145/3448016.3452764

[26] Geliang Ouyang, Jingyao Chen, Zhihe Nie, Yi Gui, Yao Wan, Hongyu Zhang, and Dongping Chen. 2025. nvAgent: Automated Data Visualization from Natural Language via Collaborative Agent Workflow. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 19534–19567. doi:10.18653/v1/2025.acl-long.960

[27] Vidya Setlur, Sarah E. Battersby, Melanie Tory, Rich Gossweiler, and Angel X. Chang. 2016. Eviza: A Natural Language Interface for Visual Analysis. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. Association for Computing Machinery, New York, NY, USA, 365–377. doi:10.1145/2984511.2984588

[28] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya Parameswaran. 2016. Effortless data exploration with zenvisage: an expressive and interactive visual analytics system. *Proc. VLDB Endow.* 10, 4 (Nov. 2016), 457–468. doi:10.14778/3025111.3025126

[29] Yuanfeng Song, Jinwei Lu, Yuanwei Song, Caleb Chen Cao, Raymond Chi-Wing Wong, and Haodi Zhang. 2025. FeVisQA: Free-Form Question Answering over Data Visualizations. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. 2726–2739. doi:10.1109/ICDE65448.2025.00205

[30] Yuanfeng Song, Jinwei Lu, and Raymond Chi-Wing Wong. 2025. CoVis: Neural and LLM-Driven Multi-Turn Interactions for Conversational Text-to-Visualization Generation: CoVis: Neural and LLM-Driven Multi-Turn... *The VLDB Journal* 35, 1 (Nov. 2025), 25 pages. doi:10.1007/s00778-025-00954-4

[31] Yuanfeng Song, Xuefang Zhao, and Raymond Chi-Wing Wong. 2024. Marrying Dialogue Systems with Data Visualization: Interactive Data Visualization Generation from Natural Language Conversations. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) *(KDD '24)*. Association for Computing Machinery, New York, NY, USA, 2733–2744. doi:10.1145/3637528.3671935

[32] Yuanfeng Song, Xuefang Zhao, Raymond Chi-Wing Wong, and Di Jiang. 2022. RGVisNet: A Hybrid Retrieval-Generation Neural Framework Towards Automatic Data Visualization Generation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) *(KDD '22)*. Association for Computing Machinery, New York, NY, USA, 1646–1655. doi:10.1145/3534678.3539330

[33] Arjun Srinivasan, Bongshin Lee, Nathalie Henry Riche, Steven M. Drucker, and Ken Hinckley. 2020. InChorus: Designing Consistent Multimodal Interactions for Data Visualization on Tablet Devices. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. doi:10.1145/3313831.3376782

[34] Arjun Srinivasan, Nikhila Nyapathy, Bongshin Lee, Steven M. Drucker, and John Stasko. 2021. Collecting and Characterizing Natural Language Utterances for Specifying Data Visualizations. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 464, 10 pages. doi:10.1145/3411764.3445400

[35] Arjun Srinivasan and John Stasko. 2018. Orko: Facilitating Multimodal Interaction for Visual Exploration and Analysis of Networks. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2018), 511–521. doi:10.1109/TVCG.2017.2745219

[36] Yuan Tian, Weiwei Cui, Dazhen Deng, Xinjing Yi, Yurun Yang, Haidong Zhang, and Yingcai Wu. 2025. ChartGPT: Leveraging LLMs to Generate Charts From Abstract Natural Language. *IEEE Transactions on Visualization and Computer Graphics* 31, 3 (2025), 1731–1745. doi:10.1109/TVCG.2024.3368621

[37] Yun Wang, Zhitao Hou, Leixian Shen, Tongshuang Wu, Jiaqi Wang, He Huang, Haidong Zhang, and Dongmei Zhang. 2023. Towards Natural Language-Based Visualization Authoring. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2023), 1222–1232. doi:10.1109/TVCG.2022.3209357

[38] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*. https://openreview.net/forum?id=uAjxFFing2

[39] Yifan Wu, Lutao Yan, Leixian Shen, Yunhai Wang, Nan Tang, and Yuyu Luo. 2024. ChartInsights: Evaluating Multimodal Large Language Models for Low-Level Chart Question Answering. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 12174–12200. doi:10.18653/v1/2024.findings-emnlp.710

[40] Yupeng Xie, Yuyu Luo, Guoliang Li, and Nan Tang. 2024. HAIChart: Human and AI Paired Visualization System. *Proc. VLDB Endow.* 17, 11 (July 2024), 3178–3191. doi:10.14778/3681954.3681992

[41] Bowen Yu and Cláudio T. Silva. 2017. VisFlow - Web-based Visualization Framework for Tabular Data with a Subset Flow Model. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 251–260. doi:10.1109/TVCG.2016.2598497

[42] Bowen Yu and Cláudio T. Silva. 2020. FlowSense: A Natural Language Interface for Visual Data Exploration within a Dataflow System. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2020), 1–11. doi:10.1109/TVCG.2019.2934668

[43] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3911–3921. doi:10.18653/v1/D18-1425

[44] Xingchen Zeng, Haichuan Lin, Yilin Ye, and Wei Zeng. 2025. Advancing Multimodal Large Language Models in Chart Question Answering with Visualization-Referenced Instruction Tuning. *IEEE Transactions on Visualization and Computer Graphics* 31, 1 (2025), 525–535. doi:10.1109/TVCG.2024.3456159

[45] Weixu Zhang, Yifei Wang, Yuanfeng Song, Victor Junqiu Wei, Yuxing Tian, Yiyan Qi, Jonathan H. Chan, Raymond Chi-Wing Wong, and Haiqin Yang. 2024. Natural Language Interfaces for Tabular Data Querying and Visualization: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 36, 11 (2024), 6699–6718. doi:10.1109/TKDE.2024.3400824

[46] Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2024. ReAcTable: Enhancing ReAct for Table Question Answering. *Proc. VLDB Endow.* 17, 8 (April 2024), 1981–1994. doi:10.14778/3659437.3659452

[47] Jun-Peng Zhu, Peng Cai, Kai Xu, Li Li, Yishen Sun, Shuai Zhou, Haihuang Su, Liu Tang, and Qi Liu. 2024. AutoTQA: Towards Autonomous Tabular Question Answering through Multi-Agent Large Language Models. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 3920–3933. doi:10.14778/3685800.3685816

Received July 2025; revised October 2025; accepted November 2025