VisInteract: Towards Dynamic Interactive Text-to-Visualization under Imperfect Queries

Wenxin Xu[1]* **Jinwei Lu**[1]* **Hwanhee Kim**[1] **Chen Jason Zhang**[1]†
Xiao-Yong Wei[1]† **Haoyang Li**[1] **Yuanfeng Song**†
[1]The Hong Kong Polytechnic University

Abstract

Real-world visualization requests are routinely ambiguous, incomplete, or factually incorrect, yet existing Text-to-Visualization (Text-to-Vis) systems assume well-specified inputs and produce charts in a single pass. When queries are imperfect, a system must *interact* with the user to recover the true intent, but no benchmark or method supports this dynamic process. We introduce **VisInteract**, a new paradigm that reframes Text-to-Vis as interaction-driven intent recovery, and **VisInteract-Bench**, to our knowledge the first benchmark for dynamic interactive Text-to-Vis, featuring controlled imperfection injection, a leakage-controlled User Agent for adaptive multi-turn feedback, and dual-perspective (code and chart) automated evaluation. On the algorithmic side, we propose **Vis-MCTS**, a method based on Monte Carlo Tree Search (MCTS) with three improvements over classical MCTS. *Progressive Widening* tames the unbounded tool-argument space in tree search, *cross-rollout memory* lets clarifications and critiques benefit the entire search tree, and *Dimension-Aware Reward Decomposition* routes scalar user feedback along data-fidelity, visual-design, and intent-alignment dimensions to resolve credit assignment across heterogeneous actions. Experiments across two LLM backbones show that Vis-MCTS consistently outperforms all Text-to-Vis baselines, improving end-to-end task success by 13.40–16.27 percentage points over the strongest interactive baseline and by more than $5\times$ over non-interactive ones. Code is available at `https://github.com/wxxv/VisInteract`.

1 Introduction

Consider a request from our benchmark (Figure 2). An analyst asks for "a nice pie chart" that ranks the top 15 California counties by K-12 enrollment "for 2025". The database covers only the 2014–2015 school year, and what the analyst actually needs is a bar chart sorted from the largest county to the smallest, since fifteen pie slices hide the ranking. Such under-specified or mistaken requests are common in real Text-to-Vis usage [1, 2]. A single-pass system can only guess. Filtering on 2025 returns nothing, and silently switching to 2014–2015 may not be what the analyst wants. Asking the analyst settles the year, and showing a chart reveals that bars are wanted.

Text-to-Vis has advanced from sequence-to-sequence translation into declarative grammars such as Vega-Lite or VQL [3] to LLM-based code generation [4–6] and agent-based pipelines [7–9], with benchmarks growing alongside [1, 10–12]. Yet all assume that the query already encodes the user's true intent, and none proactively probes what is missing, ambiguous, or incorrect (Figure 1a, b). Once this assumption is relaxed, two fundamental challenges arise.

*Equal contribution.
†Corresponding authors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

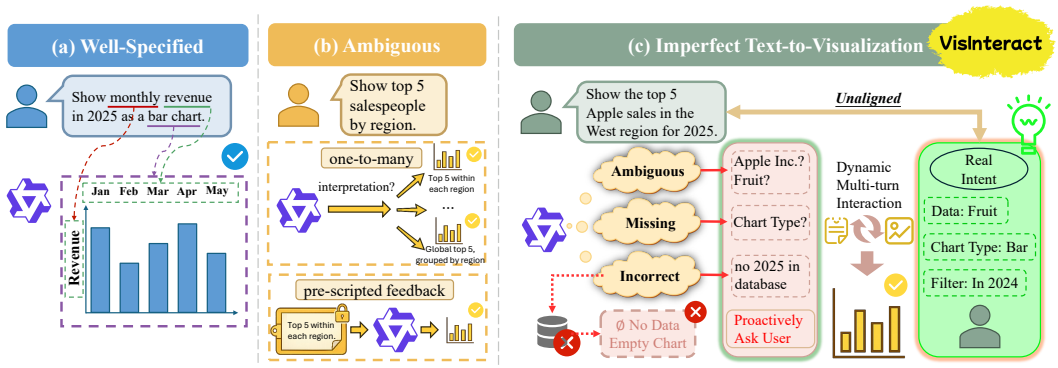


Figure 1: Comparison of Text-to-Vis paradigms. **(a)** Well-specified queries map to a single target chart. **(b)** Ambiguous queries admit multiple interpretations, which prior work handles with static mappings or pre-scripted feedback. **(c) VisInteract** recovers user intent from imperfect queries through dynamic, multi-turn, multi-modal interaction.

Challenge 1: Existing benchmarks neither model imperfect inputs nor evaluate open-ended outputs. On the input side, real-world queries are routinely *ambiguous*, *incomplete*, or *factually incorrect*, yet current benchmarks remain single-turn [10, 13], pre-scripted [14], or limited to static feedback [9] (Figure 1b). Adjacent NL2SQL benchmarks such as BIRD-INTERACT [15] demonstrate the value of dynamic clarification, but no analogous benchmark exists for visualization, which further requires *visual* feedback on chart design. On the output side, unlike NL2SQL where result-set equivalence yields a clean correctness signal, visualization is inherently open-ended, and exact-match or execution-equivalence metrics [10, 16, 17] penalize correct-but-different designs, conflating *whether* user requirements are met with *how* they are implemented.

Challenge 2: Existing methods cannot proactively explore users' true intent through dynamic interactions. Pipeline approaches [4, 5, 7] render one visualization without interaction, ambiguity-focused methods [13, 18] enumerate fixed candidates and ask at most one question to pick from them, and feedback-based variants [9] react only to pre-scripted, user-initiated edits along one trajectory. None can recover from an imperfect initial hypothesis or systematically explore the design space.

These gaps motivate **VisInteract**, a paradigm that reframes Text-to-Vis as dynamic, interaction-driven intent recovery (Figure 1c), with the following contributions:

- **VisInteract-Bench** (§3), to our knowledge the first benchmark for dynamic interactive Text-to-Vis, combining *controlled imperfection injection* of ambiguity, incompleteness, and factual errors, a *leakage-controlled User Agent* that simulates adaptive multi-turn feedback, and a *dual-perspective evaluation* in which an LLM-as-judge [19, 20] scores both the generated code and the rendered chart against user intent, avoiding over-penalization by exact-match metrics.
- **Vis-MCTS** (§4), a Monte Carlo Tree Search (MCTS) method for interactive intent recovery. Tree search lets the agent back off from an early misreading and try alternative designs, which a single trajectory cannot do. Three components adapt it. *Progressive Widening* bounds the open-ended space of tool arguments, *cross-rollout memory* lets every rollout reuse the clarifications and critiques gathered so far, and *Dimension-Aware Reward Decomposition* splits the user's score into data, design, and intent rewards so that each action is credited only for its own dimension.
- **Empirical findings** (§5), where Vis-MCTS consistently outperforms strong ReAct [21], Best-of-N, and multi-agent [7] baselines on two backbones and under matched budgets, and removing clarification, visual feedback, cross-rollout memory, or reward decomposition lowers accuracy.

2 Task Definition

Unlike SQL, where a query has a unique result set, visualization admits multiple valid designs for the same analytical intent. We therefore represent user intent not as a single target chart, but as a set of *key features* that any satisfactory visualization must exhibit.

Table 1: Comparison of Text-to-Vis benchmarks. To our knowledge, VisInteract-Bench is the first to combine dynamic interaction, multi-modal feedback, and explicitly imperfect queries. ✓ marks queries that cover only part of the imperfections (ambiguity only, or naturally under-specified queries). n/r: not reported by the source paper.

Dataset	#Tables	#Samples	Interaction Type	Interaction Modality	NL Query Imperfect	Output Format	Construction
NLV Corpus [1]	3	814	✗	✗	✓	Vega-Lite	Human
VL2NL [11]	1,981	3,962	✗	✗	✓	Vega-Lite	LLM
VisEval [12]	748	2,524	✗	✗	✗	VQL	LLM
NVBench [10]	780	25,750	✗	✗	✗	VQL	Program
NVBench 2.0 [13]	780	24,076	✗	✗	✓	VQL	LLM
MultiVis-Bench [7]	697	1,202	✗	✗	✗	Python Code	Human+LLM
Dial-NVBench [14]	780	124,449	Static	Text	✗	VQL	Program
NVBench-Feedback [9]	n/r	48,482	Static	Text	✗	VQL	LLM
VisInteract-Bench	75	1,098	Dynamic	Text + Visual	✓	Python Code	Human+LLM

Key features. Let $\mathcal{K} = \{k_1, \ldots, k_m\}$ denote the user's latent visualization intent, where each key feature $k_i = (\rho_i, d_i, \mu_i)$ pairs a type $\rho_i \in \mathcal{P}$ with a natural-language description d_i and a binary `must` flag $\mu_i \in \{0, 1\}$ indicating whether satisfying k_i is mandatory ($\mu_i = 1$) or merely preferred ($\mu_i = 0$). We write $\mathcal{K}^{\text{must}} = \{k_i \in \mathcal{K} : \mu_i = 1\}$ for the must-have subset. The type vocabulary $\mathcal{P} = \{$`mark`, `encoding`, `filter`, `aggregation`, `composition`, `interaction`$\}$ follows the canonical primitives of the Grammar of Interactive Graphics [22], jointly spanning data-level transformations and visual-level specifications, with overlays such as statistical reference lines subsumed under `composition`. Together these six categories already cover the design decisions exercised by mainstream Text-to-Vis benchmarks [12, 13].

Traditional Text-to-Vis. Given a database $\mathcal{D}$ and a well-specified query q aligned with $\mathcal{K}$, the task is a mapping $f : (\mathcal{D}, q) \mapsto \mathcal{Y}$ producing visualization $\mathcal{Y} = (c, z)$, where c is executable code and z is the corresponding rendered chart, such that $\mathcal{Y} \models \mathcal{K}^{\text{must}}$, that is, both c and z satisfy every must-have key feature. Optional key features count only toward the KF Pass Rate (§3).

VisInteract setting. In practice, the system receives an imperfect query $\tilde{q} = \pi(\mathcal{K})$, where projection π may omit, distort, or under-specify parts of $\mathcal{K}$. Since π is not injective, $\tilde{q}$ is consistent with multiple plausible key-feature sets, making intent convergence through interaction necessary. We use Ω to denote the observation process, through which the system obtains indirect evidence about $\mathcal{K}$ from textual clarification or visual feedback without ever observing $\mathcal{K}$ itself. VisInteract defines:

$$g^{\Omega} : (\mathcal{D},\, \tilde{q}) \;\mapsto\; \mathcal{Y} \quad \text{s.t.} \quad \mathcal{Y} \models \mathcal{K}^{\text{must}}.$$

Since $\mathcal{K}$ is never directly accessible, the system must decide *when* and *what* to ask through Ω, giving rise to a partially observable sequential decision problem (§4).

3 VisInteract-Bench

Table 1 summarizes the Text-to-Vis benchmark landscape. Prior datasets either assume well-specified inputs or offer only static interactions. VisInteract-Bench fills this gap with 1,098 samples over 11 BIRD Mini-Dev [23] databases (75 tables).

Sample structure. As illustrated in Figure 2, each sample is a tuple $(\tilde{q},\, \mathcal{K},\, \mathcal{Y})$ comprising the imperfect query $\tilde{q}$ exposed to the system, the latent key feature set $\mathcal{K}$, and the validated ground-truth visualization $\mathcal{Y}$. Key features $\mathcal{K}$ serve three roles simultaneously, namely 1) defining the user's true intent and only exposed to the User Agent, 2) specifying the targets for imperfection injection, and 3) providing the criteria for automated evaluation.

Construction pipeline. A five-stage pipeline builds each sample. (1) *Source data*. The BIRD Mini-Dev databases span diverse domains (sports, entertainment, education, finance, healthcare, etc.) and schema complexities. (2) *Candidate generation*. For each source instance, an LLM grounded in parsed schema and SQL semantics, and steered by a global *diversity tracker* that biases it toward under-represented chart families and imperfection templates, proposes ten structured visualization candidates, which a per-instance *diversity maximizer* then prunes to five. (3) *Ground-truth generation*. For each candidate, the LLM produces chart-aligned SQL and Altair [24] code, executed in a sandbox with bounded repair loops. Samples that fail SQL execution, chart contracts, or visual quality

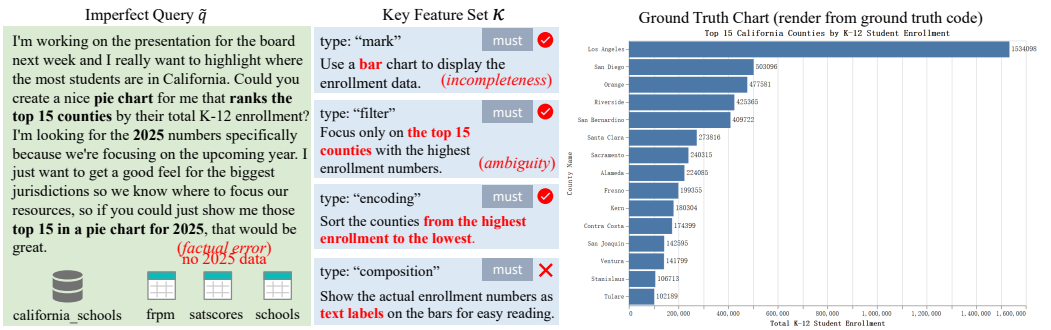


Figure 2: A VisInteract-Bench sample. The system observes only the imperfect query $\tilde{q}$. The key-feature set $\mathcal{K}$ and the ground-truth visualization $\mathcal{Y}$ stay hidden, and jointly define the true intent, the injected imperfections, and the evaluation criteria.

are discarded, and structured plus natural-language key features are extracted from the validated reference. (4) *Controlled imperfection injection*. An LLM rewrites the clear query by combining 1–3 of 14 *imperfection templates* from three categories (*ambiguity* for multiple plausible readings, *incompleteness* for critical details dropped, and *factual error* for things like a non-existent column), and records the targeted Vega-Lite paths in an *imperfection profile*. Injection occurs after validation, so every imperfect query retains a correct executable reference. (5) *Quality control*. Each candidate passes three automated checks, namely *data shape* (SQL result fits the chart contract), *spec coverage* (rendered chart matches $\geq 80\%$ of the structured key features), and *imperfection strength* (imperfect query is neither too explicit nor too vague against the imperfection profile), followed by human review on naturalness, visual quality, and key-feature faithfulness. Only samples that pass are kept.

The resulting VisInteract-Bench is a zero-shot evaluation set of 1,098 samples from 490 source BIRD questions across seven domains. These samples carry 5,147 natural-language key features in total, 4.69 per sample on average, of which 76.8% are `must` (3.60 must-have and 1.09 optional). Imperfection categories overlap: 61.1% of queries contain ambiguity, 69.1% a factual error, and 6.2% information incompleteness, averaging 1.36 categories per sample. Per-template and per-category breakdowns are in Appendix B.

Leakage-controlled User Agent. The User Agent simulates a user who knows their intent ($\mathcal{K}$, ground-truth code and chart) but never proactively reveals it. It exposes two complementary interfaces, guarded by *access gating* (which subset of $\mathcal{K}$ is visible) and a *response policy* (what the model may verbalize) to prevent ground-truth leakage. $\texttt{ask_user_text}(q_{\text{text}}) \to a_{\text{text}}$ answers a textual question after a pre-filtering step in which a scanner LLM labels which key-feature types the question pertains to, and only the matched features are exposed to the User Agent. $\texttt{ask_user_vis}(c) \to (s, m)$ executes c, shows the rendered chart alongside the ground-truth chart to a VLM, and returns up to three observations m, most important first, that describe *what is wrong* rather than *how to fix it*, with a satisfaction score $s \in [1, 10] \cap \mathbb{Z}$, exposing only visual-dimension features. Text clarification thus resolves data-level ambiguity, and visual feedback covers chart design. The response policy refuses any direct request for the ground-truth intent. Full prompts are in Appendix D.6.

Dual-judge evaluation. For each key feature $k \in \mathcal{K}$, two complementary binary LLM judges [19, 25] decide whether a submission $\mathcal{Y} = (c, z)$ satisfies k. The *Code Judge* $J_{\text{code}}(c, k) \in \{0, 1\}$ checks the predicted code c against k with line-level evidence, while the *Chart Judge* $J_{\text{chart}}(z, k) \in \{0, 1\}$ inspects the rendered chart z to verify k visually, catching code-invisible issues such as overlapping labels or mis-mapped colors. This yields three reporting perspectives:

$$\sigma_{\text{code}}(k) = J_{\text{code}}(c, k), \quad \sigma_{\text{chart}}(k) = J_{\text{chart}}(z, k), \quad \sigma_{\text{merge}}(k) = \sigma_{\text{code}}(k) \wedge \sigma_{\text{chart}}(k). \tag{1}$$

Over the n benchmark samples, we report four metrics for each perspective $\sigma(\cdot)$ in Eq. 1. **KF Pass Rate** $= \sum_i \sum_{k \in \mathcal{K}_i} \sigma(k) / \sum_i |\mathcal{K}_i|$ is the share of all key features that are satisfied. **Must-KF Pass Rate** is the same ratio over the must-have features $\mathcal{K}_i^{\text{must}}$. **Task Score** $= |\{i : \sigma(k) = 1 \, \forall k \in \mathcal{K}_i^{\text{must}}\}| / n$ is the share of samples whose must-have features are all satisfied, and it is our main metric. **Renderable Rate** is the share of submissions whose code executes and renders without error. A submission that fails to render satisfies no key feature. We validate the judges in §5.4.

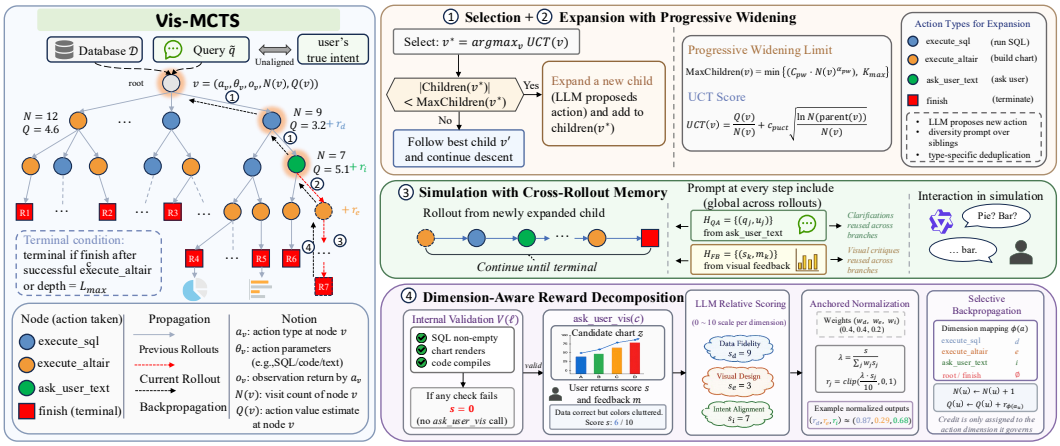


Figure 3: Overview of Vis-MCTS. Selection (1) descends by UCT, and expansion (2) adds a child while a node is below its progressive-widening cap. Simulation (3) completes the trajectory, with clarifications and critiques shared across rollouts. At a terminal node (4), the user's score is split into data, design, and intent rewards, and each ancestor receives the reward of its action type.

4 Vis-MCTS: Dynamic Interactive Text-to-Vis via Monte Carlo Tree Search

4.1 Overview

Figure 3 illustrates the complete Vis-MCTS framework. We model dynamic interactive Text-to-Vis as a partially observable sequential decision problem [26], in which the system observes an imperfect query $\tilde{q}$ and a database $\mathcal{D}$ while the true user intent $\mathcal{K}$ remains latent, and we adopt Monte Carlo Tree Search (MCTS) [27] as the underlying search backbone. We define the LLM agent's action space as $\mathcal{A} = \{\texttt{execute_sql}, \texttt{execute_altair}, \texttt{ask_user_text}, \texttt{ask_user_vis}, \texttt{finish}\}$. At each decision step t, the agent selects an action $a_t \in \mathcal{A}$ conditioned on the accumulated history $\{(a_i, o_i) | 0 < i < t\}$, invokes the corresponding tool, and receives an observation o_t. `ask_user_vis` is not a selectable action during tree search. The search framework invokes it only at valid terminal nodes to obtain a user satisfaction score s from the User Agent, which serves as the reward signal for backpropagation and the objective that Vis-MCTS maximizes.

Adapting traditional MCTS to interactive Text-to-Vis raises three challenges that existing LLM tree-search methods [28, 29] leave unaddressed: an unbounded action parameter space, the lack of cross-rollout information sharing, and incorrect credit assignment under scalar-reward backpropagation. Vis-MCTS addresses them via *Progressive Widening* (§4.3), cross-rollout memory during simulation (§4.4), and *Dimension-Aware Reward Decomposition* (§4.5), respectively.

4.2 Search Tree Structure

Each node v in the search tree is one tool-call step in a ReAct-style [21] reasoning chain:

$$v = \big(a_v,\ \theta_v,\ o_v,\ N(v),\ Q(v)\big), \tag{2}$$

where $a_v \in \mathcal{A}$ is the action, θ_v is its parameters (e.g., SQL text or Altair code), o_v is the tool observation, $N(v) \in \mathbb{N}$ is the *visit count* that counts how many rollouts have traversed v, and $Q(v) \in \mathbb{R}_{\geq 0}$ is the *cumulative reward* backpropagated through v so far. A root-to-leaf path $\tau = (v_0, v_1, \ldots, v_T)$ constitutes a complete LLM reasoning trajectory, that is a candidate solution. Children of the same parent represent alternative actions explored at that decision point.

A node v is *terminal* if $a_v = \texttt{finish}$ and the most recent `execute_altair` ancestor succeeded, or if the path depth reaches $L_{\max}$. Terminal nodes that pass an internal validation predicate $\mathcal{V}\colon v \to \{0, 1\}$, which checks that the code compiles, the chart renders, and SQL returns non-empty results, are submitted for user evaluation. Those with $\mathcal{V}(v) = 0$ receive zero reward without invoking `ask_user_vis` for further user feedback.

4.3 Selection with Progressive Widening

In classical MCTS, selection descends the tree by choosing the child with the highest Upper Confidence Bounds for Trees (UCT) [27] score, then *exhaustively* expands all legal moves at the selected node. This is feasible when the action space is discrete and small (e.g., board games), but in Vis-MCTS the parameter space Θ_a of each action is effectively unbounded, since the LLM can generate infinitely many distinct SQL queries, Altair programs, or clarification questions.

We therefore adopt *Progressive Widening* [30] (Figure 3, ①②), which caps the number of children at each node and lets it grow sublinearly with the visit count:

$$\text{MaxChildren}(v) = \min\big(\lceil C_{\text{pw}} \cdot \max(N(v), 1)^{\alpha_{\text{pw}}} \rceil,\ K_{\max}\big), \tag{3}$$

where C_{pw} controls the widening rate, α_{pw} is the growth exponent, and $K_{\max}$ is a hard cap. Counting $N(v)$ as at least 1 lets a newly created node be expanded. With the settings of §5.1, a node may have up to two children until it has been visited twice, and up to three afterwards. As a result, selection no longer relies on UCT alone but jointly considers UCT and MaxChildren. At each node v during descent, if $|\text{children}(v)| < \text{MaxChildren}(v)$, the node is *expandable* and a new child is expanded. Otherwise, its child with the highest UCT score is selected and descent continues via

$$\text{UCT}(v) = \underbrace{\frac{Q(v)}{N(v)}}_{\text{exploitation}} + \underbrace{c_{\text{puct}} \sqrt{\frac{\ln N(\text{parent}(v))}{N(v)}}}_{\text{exploration}}, \tag{4}$$

where $c_{\text{puct}} > 0$ is an exploration constant that balances the two terms.

When a node is expandable, the framework prompts the LLM for a new action within the legal subspace defined by a successor function $\Gamma \colon \mathcal{A} \to 2^{\mathcal{A}}$ that maps each action to its permissible successors (Appendix D.3). Since the LLM tends to produce similar outputs under the same context, a *diversity prompt* describes existing sibling actions to encourage novelty (Appendix D.5.2), and a type-specific *deduplication check* discards any candidate whose action type and canonicalized key argument exactly match an existing sibling (Appendix D.4). If expansion succeeds, the new child is passed to simulation. If deduplication rejects it, selection falls back to UCT descent and retries at a deeper node. This process repeats at each level until a new node is created or the tree is exhausted.

4.4 Simulation

After expansion produces a new node, simulation extends the path to a terminal state (Figure 3, ③) by repeatedly invoking the LLM for the next action, subject to the successor constraints Γ, until a terminal node is reached. Throughout this process, two global memory structures are injected into the LLM prompt to form a *cross-rollout memory*, namely (i) a QA history $\mathcal{H}_{\text{QA}} = \{(q_j, u_j)\}$ that accumulates all `ask_user_text` exchanges, where q_j is the clarification question and u_j the user's reply, and (ii) a feedback history $\mathcal{H}_{\text{FB}} = \{(s_k, m_k)\}$ that records all `ask_user_vis` results, where $s_k \in [1, 10] \cap \mathbb{Z}$ is the satisfaction score and m_k the textual critique. Since both histories are shared across *all* rollouts, a single interaction benefits the entire tree, not just the branch that generated it.

4.5 Dimension-Aware Reward Decomposition

Once simulation reaches a terminal node ℓ, Vis-MCTS obtains user feedback and decomposes it into dimension-aware rewards (Figure 3, ④). If ℓ passes the validation $\mathcal{V}$ of §4.2, the search framework calls `ask_user_vis`, and the User Agent returns a satisfaction score s and textual feedback m, which are appended to $\mathcal{H}_{\text{FB}}$. Otherwise $s = 0$.

Standard MCTS would backpropagate this scalar s uniformly to all ancestors, but a visualization trajectory interleaves SQL nodes (data retrieval), Altair nodes (chart design), and text-query nodes (intent clarification), creating a credit-assignment problem: when the user reports "data is correct but the chart type is wrong" with a low overall score, a SQL node that issued a correct query is unfairly penalized for an unrelated visual deficiency. Thus we decompose the scalar reward into three interpretable dimensions, each aligned with an action type:

Step 1: Dimension-wise relative scoring. We define a scoring function $\mathcal{R} \colon \mathcal{I} \times \mathcal{T} \times \mathcal{S} \times \mathcal{M} \to \mathcal{S}^3$, where $\mathcal{I}$ denotes the rendered chart images, $\mathcal{T}$ denotes the trajectory, $\mathcal{S}$ denotes the user scores, and $\mathcal{M}$

denotes the corresponding textual feedback. An LLM instantiates $\mathcal{R}$ to produce per-dimension scores $(s_d, s_e, s_i) = \mathcal{R}(\text{img}, \tau, s, m)$ reflecting data fidelity, visualization design, and intent alignment respectively. The prompt focuses on *relative* quality differences, which dimensions the feedback praises or criticizes, rather than absolute calibration (full prompt template in Appendix D.5.3).

Step 2: Anchored normalization. We scale the LLM-assigned scores so that their weighted combination matches the original user score:

$$\lambda = \frac{s}{\sum_{j\in\{d,e,i\}} w_j \cdot s_j}, \qquad r_j = \text{clip}\Big(\frac{\lambda \cdot s_j}{10},\ 0,\ 1\Big)\ \ \forall\, j \in \{d, e, i\}, \tag{5}$$

where (w_d, w_e, w_i) are dimension weights with $\sum_j w_j = 1$. Without clipping, $\sum_j w_j \cdot r_j = s/10$, so the LLM decides how the user's score is split across dimensions but not how large it is. Clipping, which keeps each r_j in $[0, 1]$, can only lower this sum. §5.4 works through a concrete example.

4.6 Selective Backpropagation

Given decomposed reward vector (r_d, r_e, r_i), we define a dimension mapping ϕ: $\mathcal{A} \rightarrow \{d, e, i, \emptyset\}$ that sends `execute_sql`, `execute_altair`, `ask_user_text` to d, e, i respectively, and `finish`/`root` to $\emptyset$. During backpropagation from terminal leaf ℓ, each ancestor u updates:

$$N(u) \leftarrow N(u) + 1, \qquad Q(u) \leftarrow Q(u) + r_{\phi(a_u)} \cdot \mathbf{1}[\phi(a_u) \neq \emptyset]. \tag{6}$$

Except for `root` and `finish`, which update only N, each node accumulates a Q-value drawn solely from the dimension it governs, namely r_d at `execute_sql`, r_e at `execute_altair`, and r_i at `ask_user_text`. The exploitation term $Q(v)/N(v)$ therefore reflects the node's own-dimension performance rather than a mixed average across dimensions, making useful partial trajectories less likely to be penalized for errors made in unrelated later decisions. A node is treated as exhausted only if it cannot admit new children under the current progressive-widening cap and all existing children are exhausted. Because MaxChildren(v) increases with $N(v)$, this condition is re-evaluated as visit counts change, preventing early saturation from permanently closing a branch.

After the last rollout, Vis-MCTS returns the code of the terminal node with the highest user score s, with ties going to the latest rollout. Algorithm 1 (Appendix A) gives the full procedure.

5 Experiments

5.1 Experimental Setup

Benchmark and evaluation. We evaluate all methods on the 1,098 samples of VisInteract-Bench with the dual-judge pipeline (§3), using `Qwen3.5-plus` as the judge LLM. We report KF Pass Rate, Must-KF Pass Rate, and Task Score under the Code, Chart, and Merge perspectives. Every method is run three times end to end (agent, User Agent, and judge), and we report the mean.

Baselines. *Without user interaction*, **Self-Correction LLM** generates a visualization with the LLM and repairs execution errors iteratively, and **nvAgent** [8] runs a fixed Processor, Composer, and Validator workflow originally designed for VQL, whose output is translated to Python code. *With user interaction*, **ReAct** [21] interleaves reasoning with tool calls in a single trajectory, **Best-of-N (ReAct)** runs $N = 10$ independent ReAct chains and returns the chain with the highest user score, and **MultiVis-Agent** [7] is a multi-agent system with logic-rule constraints, consisting of a Coordinator, SQL and code Generators, and a Validator. All methods use two backbones, `Qwen3.5-flash` and `Gemini-3.1-flash-lite-preview` (`Qwen` and `Gemini` below). Details are in Appendix C.

Vis-MCTS configuration. We use $N_{\text{rollout}} = 10$ rollouts, maximum trajectory depth $L_{\max} = 20$, and progressive-widening child cap $K_{\max} = 3$. Per sample, Vis-MCTS may ask at most ten clarification questions and request at most ten visual evaluations, shared across all rollouts. Each Best-of-N chain has its own budget of ten of each (Appendix C). Appendix D.2 lists all hyperparameters.

5.2 Main Results

Table 2 presents the main results. Across its cells, the standard deviation over the three runs averages 0.86 points. We highlight three findings:

Table 2: Main results on VisInteract-Bench (mean of three runs). †Non-interactive methods. ‡Interactive methods. Within each backbone, the best result is **highlighted in blue** and the second-best in light blue. Δ: gain of Vis-MCTS over the best baseline, in percentage points.

Method	Code-level			Chart-level			Merge			
	KF	Must-KF	Task	KF	Must-KF	Task	KF	Must-KF	Task	Renderable
Qwen3.5-flash										
Self-Correction LLM†	37.19	38.43	11.02	24.71	30.28	10.11	21.76	26.84	7.38	95.54
nvAgent†	11.68	14.57	2.19	9.81	12.02	2.19	8.08	10.12	1.64	66.67
MultiVis-Agent‡	41.67	43.03	15.19	29.00	35.47	12.71	25.97	32.07	9.48	94.35
ReAct‡	66.62	69.72	41.17	56.25	66.10	38.80	50.83	60.38	29.05	94.72
Best-of-N (ReAct)‡	74.22	77.64	50.00	64.35	75.36	46.81	59.68	70.40	38.06	99.73
Vis-MCTS	**81.06**	**84.92**	**61.66**	**73.87**	**85.96**	**64.39**	**69.11**	**81.05**	**51.46**	**100.00**
Δ	↑6.84	↑7.28	↑11.66	↑9.52	↑10.60	↑17.58	↑9.43	↑10.65	↑13.40	↑0.27
Gemini-3.1-flash-lite-preview										
Self-Correction LLM†	41.29	41.59	13.93	30.00	36.38	14.21	26.00	31.85	10.66	97.63
nvAgent†	15.43	19.45	3.28	13.19	16.32	3.28	10.71	13.53	2.00	81.06
MultiVis-Agent‡	56.05	56.99	30.33	44.96	53.45	29.69	40.41	48.14	22.13	98.54
ReAct‡	68.64	71.46	47.81	60.79	71.03	47.72	56.50	66.25	39.62	95.17
Best-of-N (ReAct)‡	75.43	76.35	54.79	62.92	71.51	53.79	57.90	70.78	40.92	99.73
Vis-MCTS	**83.06**	**86.61**	**67.21**	**77.32**	**87.98**	**69.67**	**72.77**	**81.97**	**57.19**	**100.00**
Δ	↑7.63	↑10.26	↑12.42	↑14.40	↑16.47	↑15.88	↑14.87	↑11.19	↑16.27	↑0.27

Vis-MCTS is best on every metric. Under both backbones, Vis-MCTS reaches Code/Chart/Merge Task Scores of 61.66% / 64.39% / 51.46% on Qwen and 67.21% / 69.67% / 57.19% on Gemini, with gains of 11.66 / 17.58 / 13.40 and 12.42 / 15.88 / 16.27 points over the strongest baseline on each perspective. Because a feature counts toward Merge only when both judges accept it, the gains on Merge show that Vis-MCTS improves data correctness and visual design together rather than one at the expense of the other. Vis-MCTS also renders a valid chart for every sample on both backbones. Best-of-N comes close (99.73%), while ReAct fails on about 5% of samples (94.72% / 95.17% Renderable) and nvAgent on up to a third (66.67% / 81.06%). The accuracy gains of Vis-MCTS therefore do not come from skipping hard samples.

Interaction is essential. Non-interactive methods do not recover intent once the query is imperfect. Self-Correction LLM, the stronger of the two, reaches only 7.38% and 10.66% Merge Task Score on Qwen and Gemini, against 51.46% and 57.19% for Vis-MCTS, a gap of more than 5×, and nvAgent falls further, to 1.64% and 2.00%. A further round of reasoning or execution repair stays inside the initial misreading, because the constraint the query omitted is never observed. Removing either interaction channel from Vis-MCTS confirms this (§5.3).

Search with shared memory outperforms repeated sampling. With the same ten rollouts, Vis-MCTS exceeds Best-of-N (ReAct) by 13.40 and 16.27 points of Merge Task Score on Qwen and Gemini (51.46% / 57.19% vs. 38.06% / 40.92%), although each Best-of-N chain may ask up to ten questions and request up to ten visual checks on its own, ten times the cap that Vis-MCTS shares across all its rollouts. Best-of-N's own gain over a single ReAct chain is smaller and inconsistent across backbones (+9.01 points on Qwen, +1.30 on Gemini). The gap over Best-of-N combines tree search with the memory shared across rollouts, and §5.3 separates the two under matched budgets.

5.3 Ablation Studies

To isolate each Vis-MCTS component, we evaluate four variants that disable the two interaction channels (textual and visual) and two of the algorithmic components that distinguish Vis-MCTS from standard MCTS. **w/o Textual Interaction.** Remove `ask_user_text` from the action space, so the agent must recover intent through code execution and visual critiques alone. **w/o Visual Feedback.** Disable `ask_user_vis`. No terminal reward is then obtained, so $Q(v) \equiv 0$, UCT reduces to pure exploration, and the variant returns its most-visited valid terminal chart. **w/o Cross-Rollout Memory.** Reset $\mathcal{H}_{\text{QA}}$ and $\mathcal{H}_{\text{FB}}$ at each rollout so clarifications and critiques do not persist across rollouts. **w/o Reward Decomposition.** Replace Eq. 5 with uniform backpropagation of $s/10$ to all ancestors.

Table 3: Ablation study on Vis-MCTS components with `Qwen3.5-flash` as the backbone.

Variant	Code-level			Chart-level			Merge			Renderable
	KF	Must-KF	Task	KF	Must-KF	Task	KF	Must-KF	Task	
Vis-MCTS (full)	**81.06**	**84.92**	**61.66**	**73.87**	**85.96**	**64.39**	**69.11**	**81.05**	**51.46**	**100.00**
- w/o Textual Interaction	76.24	79.69	51.82	69.09	81.05	54.64	63.55	74.93	44.08	99.82
- w/o Visual Feedback	64.58	68.63	33.42	55.31	66.20	32.79	49.99	60.26	24.24	99.73
- w/o Cross-Rollout Memory	68.23	72.88	40.07	63.14	75.77	43.99	57.30	69.14	34.06	99.91
- w/o Reward Decomposition	73.77	77.78	50.27	68.49	77.50	51.91	61.75	71.22	43.26	99.82

Removing any component lowers every metric in Table 3. *Visual Feedback* supplies the terminal reward signal, and removing it collapses UCT to blind exploration and lowers Merge Task Score by 27.22 points. *Textual Interaction* resolves non-visual intent ambiguities (e.g., filters, aggregations) that visual critiques cannot expose, and removing it lowers Merge Task Score by 7.38 points. Without *Cross-Rollout Memory*, each rollout must rediscover what earlier rollouts already learned, and Merge Task Score falls by 17.40 points to 34.06%, below Best-of-N (38.06%). Without *Reward Decomposition*, a correct query is penalized together with a wrong chart under uniform backpropagation, and Merge Task Score falls by 8.20 points.

Matched budgets. An equal number of rollouts does not equalize time, tokens, or user calls. Under a common per-sample cap on wall-clock time, tokens, and User Agent calls (Appendix E.2), Merge Task Score is 29.05% for ReAct, 37.25% for Best-of-N, 42.71% for a Shared-Memory Best-of-N that passes earlier clarifications and critiques to later chains, and 47.63% for Vis-MCTS. Of the 10.38 points between Best-of-N and Vis-MCTS, shared memory thus accounts for about half and the search tree for the remaining 4.92. Replacing the Qwen User Agent with Gemini leaves the ranking of the interactive methods unchanged (Appendix E.2).

5.4 Analysis

Hyperparameter sensitivity. In single runs, Merge Task Score rises with the number of rollouts N_{rollout} and the maximum trajectory depth $L_{\max}$, while runtime grows sub-linearly (Appendix E.1). A separate sweep over the reward weights (w_d, w_e, w_i) across uniform weights and three settings that each favor one dimension keeps Merge Task Score within 1.64 points of the default (Table 4), so the dimension-aware decomposition does not require fine-tuned weights.

Table 4: Reward-weight sensitivity (Task Score) on `Qwen3.5-flash`. All other experiments use the default $(0.4, 0.4, 0.2)$.

Configuration	w_d	w_e	w_i	Code	Chart	Merge
Default	0.40	0.40	0.20	**61.66**	**64.39**	**51.46**
Uniform	0.33	0.33	0.33	61.02	63.75	50.91
Data-heavy	0.60	0.20	0.20	61.29	63.02	50.55
Design-heavy	0.20	0.60	0.20	60.38	64.03	50.27
Intent-heavy	0.20	0.20	0.60	60.11	62.84	49.82

Case study. Figure 6 (Appendix E.3) follows the request from the introduction (Figure 2), which asks for a pie chart of 2025 enrollment although the database covers only 2014–2015. The search confirms the year with `ask_user_text`, and the answer is shared with all later rollouts. At a later terminal node the user returns $s = 2$ and the feedback "I expected to see a bar chart that ranks the counties by size from largest to smallest." Dimension-wise scores $(s_d, s_e, s_i) = (10, 3, 2)$ then normalize, under the default weights, to $(r_d, r_e, r_i) = (0.357, 0.107, 0.071)$. The SQL node that retrieved the correct data thus receives the most credit, followed by the Altair node that drew the pie chart and the clarification node, which settled the year but not the chart type. Uniform backpropagation would instead assign 0.20 to every ancestor.

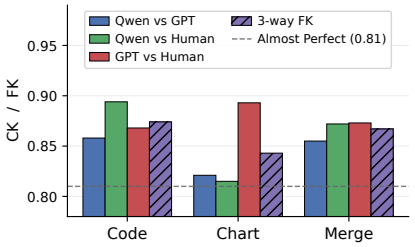


Figure 4: Inter-judge agreement on VisInteract-Bench. CK denotes Cohen's Kappa coefficient κ, and FK denotes Fleiss' Kappa κ_F.

Judge reliability. To test whether the ranking in Table 2 depends on the judge, we compare our judge LLM (`Qwen3.5-plus`, abbrev. `Qwen`) with a second judge from a different family (`GPT-5.4-mini`, abbrev. `GPT`) and with human experts on stratified subsets of the benchmark (Appendix E.4). As shown in Figure 4, all pairwise CK and the 3-way FK exceed the 0.81 "almost perfect" threshold [31],

and our `Qwen` judge agrees with humans as closely as `GPT` does (mean CK $= 0.860$ vs. 0.878). Adding `Gemini-3.5-flash-lite` as a third LLM judge keeps Fleiss' κ across the three model families at 0.845, 0.824, and 0.840 for Code, Chart, and Merge.

6 Related Work

Text-to-Vis and interactive NL interfaces. NVBench [10] established the first large-scale Text-to-Vis benchmark, with subsequent efforts expanding the scope via human-authored utterances [1], LLM-generated query–chart pairs [11, 12], multi-chart reasoning [7], robustness to paraphrased queries and schema variants [16, 18], and free-form question answering over visualizations [32]. More recent benchmarks shift toward interaction. NVBench 2.0 [13] injects ambiguity into clear seed queries but remains single-turn and covers ambiguity only, while Dial-NVBench [14, 33], NVBench-Feedback [9], and progressive agents checked by verification rules [34] use multi-turn or feedback-based exchanges that are pre-scripted and do not adapt to system output. Methodologically, approaches have evolved from neural translation [3] to LLM-based step-wise reasoning [4], code generation [5, 6], concept-driven authoring [35], and clarification of the most uncertain query clause with a single question [18]. Natural language interfaces to databases have also been extended from SQL to NoSQL databases [36–38], and in the parallel NL2SQL domain, MISP [39] and BIRD-INTERACT [15] pioneered dynamic clarification for resolving ambiguities. VisInteract-Bench extends this paradigm of dynamic clarification from SQL text to *rendered charts* through controlled imperfection injection and leakage-controlled multi-modal user feedback, evaluating whether a system can recover intent.

LLM agents and tree search for reasoning. Single-chain agents such as ReAct [21], Reflexion [40], and CodeAct [41] interleave reasoning with acting but cannot recover from early mistakes. Tree-structured reasoning addresses this limitation. Tree-of-Thoughts [28] and LATS [29] extend chain-of-thought with search, while RAP [42], LLaMA-Berry [43], and rStar [44] combine MCTS with LLMs for planning and self-refinement. These methods, however, rely on LLM self-evaluation as the reward signal and apply uniform backpropagation. Vis-MCTS differs in three respects. Progressive Widening [30] handles the open-ended parameter space of LLM tool calls, cross-rollout memory shares the user's clarifications and critiques across rollouts instead of the model's own reflections, and Dimension-Aware Reward Decomposition replaces uniform backpropagation. Reducing the human effort needed to collect labels or data is a long-standing goal of interactive multimedia systems, from active learning that balances exploration and exploitation in interactive video retrieval [45] to automated data collection for large-scale classroom analysis [46]. The shared memory serves the same goal, since one clarification informs every later rollout.

7 Conclusion

We introduced **VisInteract**, a paradigm that reframes Text-to-Vis as dynamic, interaction-driven intent recovery from imperfect queries. VisInteract-Bench provides a benchmark with controlled imperfections, a leakage-controlled User Agent, and dual-perspective automated evaluation. Vis-MCTS adapts Monte Carlo Tree Search to Text-to-Vis via Progressive Widening for open-ended action parameter spaces, cross-rollout memory so clarifications and critiques benefit the entire search tree, and Dimension-Aware Reward Decomposition for credit assignment across heterogeneous action types. Experiments on two LLM backbones show that Vis-MCTS consistently outperforms both interactive and non-interactive baselines, and that removing clarification, visual feedback, cross-rollout memory, or reward decomposition lowers accuracy.

Limitations. Our evaluation relies on LLM-based judges and a simulated User Agent. Judge agreement with human experts is high, but the User Agent knows a fixed intent and answers more consistently than real users, who may be unsure of their goals or change them. VisInteract-Bench covers 11 BIRD Mini-Dev databases and Altair, with imperfections injected from 14 templates, so other domains, grammars, and naturally occurring imperfections remain untested. Vis-MCTS also spends more computation than single-pass generation, about 20 minutes per sample with default settings, although it stays ahead when time, tokens, and user calls are capped equally.

Acknowledgments and Disclosure of Funding

This work was supported in part by the NSFC/RGC Joint Research Scheme (N_PolyU5179/25), the Hong Kong Research Grants Council (PolyU25600624), the Innovation and Technology Fund (ITS/052/23MX and PRP/009/22FX), industrial sponsors, and the Research Matching Grant Scheme (P0045948, P0048183, P0048191, P0046453, and P0060272). This work was also supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62372314, and HK PolyU Research Grant (No. A0064486). The experimental part of this work was supported by the Centre for Large AI Models (CLAIM) of The Hong Kong Polytechnic University. This work was also supported by Guangdong Basic and Applied Basic Research Foundation 2026A1515010227, PolyU TDG25-28/TBP/11-IICA.

References

[1] Arjun Srinivasan, Nikhila Nyapathy, Bongshin Lee, Steven M. Drucker, and John Stasko. Collecting and characterizing natural language utterances for specifying data visualizations. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–10. ACM, 2021. doi: 10.1145/3411764.3445400.

[2] Arpit Narechania, Arjun Srinivasan, and John Stasko. NL4DV: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):369–379, 2021. doi: 10.1109/tvcg.2020.3030378.

[3] Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):217–226, 2022. doi: 10.1109/tvcg.2021.3114848.

[4] Yuan Tian, Weiwei Cui, Dazhen Deng, Xinjing Yi, Yurun Yang, Haidong Zhang, and Yingcai Wu. ChartGPT: Leveraging LLMs to generate charts from abstract natural language. *IEEE Transactions on Visualization and Computer Graphics*, 31(3):1731–1745, 2025. doi: 10.1109/tvcg.2024.3368621.

[5] Victor Dibia. 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)*, pages 113–126. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-demo.11.

[6] Paula Maddigan and Teo Susnjak. Chat2VIS: Generating data visualizations via natural language using ChatGPT, Codex and GPT-3 large language models. *IEEE Access*, 11:45181–45193, 2023. doi: 10.1109/access.2023.3274199.

[7] Jinwei Lu, Yuanfeng Song, Chen Zhang, and Raymond Chi-Wing Wong. MultiVis-Agent: A multi-agent framework with logic rules for reliable and comprehensive cross-modal data visualization. *Proceedings of the ACM on Management of Data*, 4(1):56:1–56:25, 2026. doi: 10.1145/3786670.

[8] Geliang Ouyang, Jingyao Chen, Zhihe Nie, Yi Gui, Yao Wan, Hongyu Zhang, and Dongping Chen. 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)*, pages 19534–19567. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.960.

[9] Xubang Xiong, Raymond Chi-Wing Wong, and Yuanfeng Song. Interactive text-to-visualization: Refining visualization outputs through natural language user feedback. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pages 3571–3581. ACM, 2025. doi: 10.1145/3746252.3761375.

[10] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1235–1247. ACM, 2021. doi: 10.1145/3448016.3457261.

[11] Hyung-Kwon Ko, Hyeon Jeon, Gwanmo Park, Dae Hyun Kim, Nam Wook Kim, Juho Kim, and Jinwook Seo. Natural language dataset generation framework for visualizations powered by large language models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–22. ACM, 2024. doi: 10.1145/3613904.3642943.

[12] Nan Chen, Yuge Zhang, Jiahang Xu, Kan Ren, and Yuqing Yang. VisEval: A benchmark for data visualization in the era of large language models. *IEEE Transactions on Visualization and Computer Graphics*, 31(1):1301–1311, 2025. doi: 10.1109/tvcg.2024.3456320.

[13] Tianqi Luo, Chuhan Huang, Leixian Shen, Boyan Li, Shuyu Shen, Wei Zeng, Nan Tang, and Yuyu Luo. nvBench 2.0: Resolving ambiguity in text-to-visualization through stepwise reasoning. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-4172.

[14] Yuanfeng Song, Xuefang Zhao, and Raymond Chi-Wing Wong. 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*, pages 2733–2744. ACM, 2024. doi: 10.1145/3637528.3671935.

[15] Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, Linheng Han, Edward Alexander, Xintong Zhu, Rui Qin, Ruihan Yu, Yiyao Jin, Feige Zhou, Weihao Zhong, Yun Chen, Hongyu Liu, Chenhao Ma, Fatma Ozcan, Yannis Papakonstantinou, and Reynold Cheng. BIRD-INTERACT: Re-imagining text-to-SQL evaluation via lens of dynamic interactions. In *International Conference on Learning Representations*, volume 2026, pages 44285–44333, 2026.

[16] Jinwei Lu, Yuanfeng Song, Haodi Zhang, Chen Jason Zhang, Kaishun Wu, and Raymond Chi-Wing Wong. Towards robustness of text-to-visualization translation against lexical and phrasal variability. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pages 793–806. IEEE, 2025. doi: 10.1109/icde65448.2025.00065.

[17] Shuaimin Li, Xuanang Chen, Yuanfeng Song, Yunze Song, Chen Jason Zhang, Fei Hao, and Lei Chen. Prompt4Vis: prompting large language models with example mining for tabular data visualization. *The VLDB Journal*, 34(4):38, 2025. doi: 10.1007/s00778-025-00912-0.

[18] Jinwei Lu, Wenxin Xu, Yuanfeng Song, Haodi Zhang, Chen Jason Zhang, Kaishun Wu, Xiaoyong Wei, and Raymond Chi-Wing Wong. Towards robust and interactive text-to-visualization translation: enhancing model robustness and mitigating ambiguity in natural language queries. *The VLDB Journal*, 35(5):49, 2026. doi: 10.1007/s00778-026-00992-6.

[19] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc., 2023. doi: 10.52202/075280-2020.

[20] Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4334–4353. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.248.

[21] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[22] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-Lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017. doi: 10.1109/tvcg.2016.2599030.

[23] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? A BIg

bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems*, volume 36, pages 42330–42357. Curran Associates, Inc., 2023. doi: 10.52202/075280-1835.

[24] Jacob VanderPlas, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. Altair: Interactive statistical visualizations for Python. *Journal of Open Source Software*, 3(32):1057, 2018. doi: 10.21105/joss.01057.

[25] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.153.

[26] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998. doi: 10.1016/s0004-3702(98)00023-x.

[27] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. doi: 10.1109/tciaig.2012.2186810.

[28] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc., 2023. doi: 10.52202/075280-0517.

[29] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 62138–62160. PMLR, 2024.

[30] Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In *Learning and Intelligent Optimization*, pages 433–445. Springer Berlin Heidelberg, 2011.

[31] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977. doi: 10.2307/2529310.

[32] Yuanfeng Song, Jinwei Lu, Yuanwei Song, Caleb Chen Cao, Raymond Chi-Wing Wong, and Haodi Zhang. FeVisQA: Free-form question answering over data visualizations. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pages 2726–2739. IEEE, 2025. doi: 10.1109/icde65448.2025.00205.

[33] Yuanfeng Song, Jinwei Lu, and Raymond Chi-Wing Wong. CoVis: Neural and LLM-driven multi-turn interactions for conversational text-to-visualization generation. *The VLDB Journal*, 35(1):3, 2026. doi: 10.1007/s00778-025-00954-4.

[34] Wenxin Xu, Chen Jason Zhang, Xiaoyong Wei, Haoyang Li, Hwanhee Kim, Yuanfeng Song, and Raymond Chi-Wing Wong. Towards reliable agentic progressive text-to-visualization with verification rules. *arXiv preprint arXiv:2605.29692*, 2026.

[35] Chenglong Wang, John Thompson, and Bongshin Lee. Data Formulator: AI-powered concept-driven visualization authoring. *IEEE Transactions on Visualization and Computer Graphics*, 30 (1):1128–1138, 2024. doi: 10.1109/TVCG.2023.3326585.

[36] Zhiqian Qin, Yuanfeng Song, Jinwei Lu, Yuanwei Song, Shuaimin Li, and Chen Jason Zhang. MultiTEND: A multilingual benchmark for natural language to NoSQL query translation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24632–24657. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.1265.

[37] Jinwei Lu, Jiawei Lu, Chen Jason Zhang, Zhiqian Qin, Yuanfeng Song, Haodi Zhang, and Raymond Chi-Wing Wong. Bridging the gap: Enabling natural language queries for NoSQL databases through text-to-NoSQL translation. In *2027 IEEE 43rd International Conference on Data Engineering (ICDE)*. IEEE, 2027.

[38] Jinwei Lu, Zhiqian Qin, Chen Zhang, Haodi Zhang, Yuanfeng Song, and Raymond Chi-Wing Wong. QueryCraft: A natural language-driven NoSQL database querying system powered by large language models. *Proceedings of the VLDB Endowment*, 19(12):4754–4757, 2026. doi: 10.14778/3827998.3828114.

[39] Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. Model-based interactive semantic parsing: A unified framework and a text-to-SQL case study. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5447–5458. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1547.

[40] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377.

[41] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 50208–50232. PMLR, 2024.

[42] Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.507.

[43] Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, Wanli Ouyang, and Dongzhan Zhou. LLaMA-berry: Pairwise optimization for olympiad-level mathematical reasoning via O1-like Monte Carlo tree search. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7315–7337. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.375.

[44] Zhenting Qi, Mingyuan MA, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller LLMs stronger problem-solver. In *International Conference on Learning Representations*, volume 2025, pages 20788–20807, 2025.

[45] Xiao-Yong Wei and Zhen-Qun Yang. Coaching the exploration and exploitation in active learning for interactive video retrieval. *IEEE Transactions on Image Processing*, 22(3):955–968, 2013. doi: 10.1109/tip.2012.2222902.

[46] Xiao-Yong Wei and Zhen-Qun Yang. Mining in-class social networks for large-scale pedagogical analysis. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 639–648. ACM, 2012. doi: 10.1145/2393347.2393436.

A Algorithm Pseudocode

Algorithm 1 Vis-MCTS: Dynamic Interactive Text-to-Vis via Monte Carlo Tree Search

Input: Imperfect query $\tilde{q}$, database $\mathcal{D}$, and User Agent $\mathcal{U}$.
Output: Altair code c^*.

Create root v_0 from $(\tilde{q}, \mathcal{D})$ with $N(v_0)=0,\ Q(v_0)=0$
Initialize global memories $\mathcal{H}_{\text{QA}} \leftarrow \varnothing$, $\mathcal{H}_{\text{FB}} \leftarrow \varnothing$
for $i = 1$ **to** N_{rollout} **do**
  // (1) Selection with Progressive Widening (§4.3)
  $v \leftarrow v_0$
  while v is not terminal and not exhausted **do**
    if $|\text{children}(v)| < \text{MaxChildren}(v)$ **then**
      Sample $a, \theta_a \sim \text{LLM}(\cdot \mid v, \Gamma(a_v);\ \mathcal{H}_{\text{QA}},\ \mathcal{H}_{\text{FB}})$ (Eq. 3)
      if $\text{Dedup}(a(\theta_a),\ \text{children}(v))$ **then**
        $v' \leftarrow \text{Expand}(v, a(\theta_a))$; $v \leftarrow v'$
        break {go to Simulation}
      end if
    end if
    $v \leftarrow \arg\max_{c \in \text{children}(v),\ c \text{ not exhausted}} \text{UCT}(c)$ (Eq. 4)
  end while
  // (2) Simulation (§4.4)
  while v is not terminal and $\text{depth}(v) < L_{\text{max}}$ **do**
    Sample $a, \theta_a \sim \text{LLM}(\cdot \mid v, \Gamma(a_v);\ \mathcal{H}_{\text{QA}},\ \mathcal{H}_{\text{FB}})$
    $v \leftarrow \text{Expand}(v, a(\theta_a))$
    if $a =$ `ask_user_text` **then**
      $\mathcal{H}_{\text{QA}} \leftarrow \mathcal{H}_{\text{QA}} \cup \{(q_v, u_v)\}$
    end if
  end while
  $\ell \leftarrow v$
  // (3) Dimension-aware reward decomposition (§4.5)
  if $\mathcal{V}(\ell) = 1$ **then**
    $(s, m) \leftarrow \mathcal{U}.\texttt{ask_user_vis}(\text{code}(\ell))$
    $\mathcal{H}_{\text{FB}} \leftarrow \mathcal{H}_{\text{FB}} \cup \{(s, m)\}$
    $(s_d, s_e, s_i) \leftarrow \mathcal{R}(\text{img}_\ell,\ \tau_\ell,\ s,\ m)$ {LLM dimension-wise relative scoring}
    $(r_d, r_e, r_i) \leftarrow \text{AnchoredNormalize}\big(s, s_d, s_e, s_i;\ w_d, w_e, w_i\big)$ (Eq. 5)
  else
    $s \leftarrow 0$; $(r_d, r_e, r_i) \leftarrow (0, 0, 0)$
  end if
  // (4) Selective backpropagation (§4.6)
  for all v on the path from v_0 to ℓ **do**
    $N(v)$ += 1; **if** $\phi(a_v) \neq \emptyset$ **then** $Q(v)$ += $r_{\phi(a_v)}$ (Eq. 6)
  end for
  Propagate the *exhausted* flag upward: v becomes exhausted iff all children are exhausted **and** $|\text{children}(v)| \geq \text{MaxChildren}(v)$
end for
return $c^* \leftarrow \text{code}\big(\arg\max_{\ell:\, \mathcal{V}(\ell)=1}\ s_\ell\big)$

B Benchmark Statistics

VisInteract-Bench contains 1,098 samples derived from 490 source BIRD questions across the 11 databases retained in Stage 1, spanning seven broad real-world domains (Sports, Entertainment, Education, Finance, Healthcare, Q&A Community, Scientific). In aggregate, the natural-language key-feature set $\mathcal{K}$ contains 5,147 features (3,953 `must` + 1,194 optional), and within each source BIRD question the diversity maximizer of Stage 2 emits up to five structurally distinct candidates. After Stage 5 quality control, 76 source questions (15.5%) survive with a single candidate, 222 (45.3%) with two, 191 (39.0%) with three, and 1 with five, giving an average of 2.24 benchmark samples per source question.

Chart variety. Samples are realized through the curated catalog of 133 Altair chart templates organized into 11 categories (§3). All 133 templates are instantiated by at least one sample, covering all 11 categories. Per-category counts are reported in Tab. 5. Bar-family charts dominate (25.6%), but the long tail is broad, with *distributions*, *interactive charts*, and *line charts* each contributing 11–13%, while specialized families such as circular plots, area charts, and tables remain represented at the 4–6% level.

Table 5: Sample distribution by chart category. Templates are drawn from the Stage 2 Altair catalog (§3), and all 133 catalog templates appear in at least one sample. Per-category template counts sum to 140 rather than 133 because 7 templates straddle two categories (e.g., `simple_bar_chart` is classified as both *Bar Charts* and *Simple Charts* depending on its usage context).

Category	# samples	Percentage (%)	# distinct templates
Bar Charts	281	25.6	31
Distributions	141	12.8	20
Interactive Charts	132	12.0	22
Line Charts	122	11.1	19
Scatter Plots	84	7.7	9
Advanced Calculations	73	6.6	10
Circular Plots	61	5.6	6
Simple Charts	57	5.2	8
Uncertainties & Trends	52	4.7	6
Area Charts	50	4.6	5
Tables	45	4.1	4
Total	**1,098**	**100.0**	133 distinct

Imperfection distribution. Each imperfect query is generated from 1–3 of the 14 imperfection templates in §3, organized into its three user-facing categories. Across the 1,098 samples, 61.1% activate at least one *Ambiguity* template (DATA 22.7%, VISUALIZATION 38.4%), 6.2% activate *Information Incompleteness*, and 69.1% activate *Factual Error*. The categories overlap because Stage 4 may compose templates drawn from different categories in a single rewrite. Each sample carries on average 1.36 distinct imperfection categories (1,498 category occurrences in total). Per-template counts are listed in Tab. 6, where `vis_data_incompatible`, `data_range_mismatch`, and `encoding_channel_missing` are the three most common templates, jointly responsible for over half of all template occurrences.

Table 6: Per-template usage, grouped by the four `ambiguity_type` labels emitted by the Stage 4 injector and aggregated into the three user-facing categories of §3 (*Ambiguity* = DATA ∪ VISUALIZATION).

User-facing category	Injector type	Template	# occurrences
Ambiguity	DATA	`aggregation_missing`	179
		`time_granularity_missing`	139
		`topk_missing`	35
		`filter_scope_missing`	22
	VISUALIZATION	`chart_type_missing`	76
		`encoding_channel_missing`	421
		`axis_assignment_missing`	66
Information incompleteness	INFO_COMPLETION	`bucket_definition_needed`	43
		`grouping_dimension_missing`	26
		`comparison_baseline_missing`	17
Factual error	ERROR_CORRECTION	`data_range_mismatch`	559
		`field_name_error`	301
		`vis_data_incompatible`	681
		`aggregation_conflict`	169

Key-feature taxonomy. Each sample carries on average 4.69 natural-language key features (median 5, range $[3, 6]$), of which 76.8% are flagged `must` (3.60 `must` + 1.09 optional per sample on average). The full per-type breakdown is reported in Tab. 7. `encoding` dominates (34.7%) since axis/color/-

size mappings are the most numerous design decisions, followed by `mark` (23.2%), `interaction` (15.8%), and `aggregation` (11.2%), while `composition` (10.4%, including statistical-overlay sub-features) and `filter` (4.7%) account for the remainder. At the sample level, 30.8% of samples carry exactly 4 key features and 68.0% carry 5 (0.5% have 3, 0.6% have 6), while 42.5% have 3 `must`-features and 50.9% have 4. Moreover, 52.1% of samples span four distinct key-feature types and 13.5% span five, ensuring that submissions are graded on a representative cross-section of the design space rather than on a single dimension. As expected, `mark` and `encoding` appear in essentially every sample, while `interaction` is concentrated in samples whose chart type comes from the *Interactive Charts* category. This balance is the setting that the dimension-aware reward decomposition of Vis-MCTS (§4.5) targets, routing user feedback to the action type responsible for each feature group.

Table 7: Key-feature counts by type, computed over the natural-language key-feature set $\mathcal{K}$ of all 1,098 samples. `overlay_stat_line` (statistical reference lines) is reported as a sub-row of `composition` for transparency. The right-most column gives the average number of features of each type per sample.

Type	# features	Percentage (%)	per sample
`encoding`	1,787	34.7	1.63
`mark`	1,194	23.2	1.09
`interaction`	811	15.8	0.74
`aggregation`	577	11.2	0.53
`composition` (incl. `overlay_stat_line`)	535	10.4	0.49
of which `overlay_stat_line`	82	1.6	0.07
`filter`	243	4.7	0.22
Total	**5,147**	**100.0**	**4.69**
of which `must`=true	3,953	76.8	3.60
of which optional	1,194	23.2	1.09

Difficulty and domain coverage. The injector tags each sample with an aggregate difficulty (the maximum over its chosen templates), giving `easy` 9.3% (102), `medium` 77.0% (845), and `hard` 13.7% (151) of samples. Each sample is additionally annotated with a single preferred clarification interface. For 81.3% of samples the imperfection is best resolved by a textual clarification (`ask_user_text`, for data-side decisions on filters, aggregations, and field references), and for 18.7% by visual feedback on a rendered chart (`ask_user_vis`, for chart-type, layout, and encoding ambiguities). The 11 source databases span seven broad domains, with no single domain accounting for more than a quarter of the benchmark. Per-domain database and sample counts are listed in Tab. 8.

Table 8: Sample distribution by source-database domain.

Domain	# DBs	# samples	Percentage (%)	Databases (BIRD ID)
Sports	2	250	22.8	`formula_1`, `european_football_2`
Entertainment	2	228	20.8	`superhero`, `card_games`
Education	2	168	15.3	`student_club`, `california_schools`
Finance	2	143	13.0	`financial`, `debit_card_specializing`
Healthcare	1	114	10.4	`thrombosis_prediction`
Q&A Community	1	107	9.7	`codebase_community`
Scientific	1	88	8.0	`toxicology`
Total	**11**	**1,098**	**100.0**	

C Baseline Details

Self-Correction LLM. Given the imperfect query and the database schema, the LLM directly emits a self-contained Python script that queries the SQLite database and renders an Altair chart. The script is executed in a sandbox. On failure, the traceback is fed back and the LLM regenerates the full script, for up to $R_{\max} = 10$ rounds. No SQL tool, chart feedback, or user-interaction channel is exposed, so the agent can only repair runtime errors, not intent mismatches.

nvAgent. We run nvAgent unchanged, preserving its original three-role workflow (Processor $\rightarrow$ Composer $\rightarrow$ Validator) and VQL output. The generated VQL is translated to Python code via nvAgent's own post-processing so results are comparable under our evaluation pipeline. As a non-interactive baseline, no clarification or visual-feedback tool is added.

ReAct. The ReAct agent uses the same tool set as Vis-MCTS (`execute_sql`, `execute_altair`, `ask_user_text`, `ask_user_vis`, `finish`) and interacts with the same User Agent, but operates as a single reasoning chain. It shares Vis-MCTS's action-successor constraints Γ (Appendix D.3), and its chain depth is capped at 20 steps, equal to the maximum depth of a Vis-MCTS rollout. Unlike Vis-MCTS, where the search framework calls `ask_user_vis` only at terminal nodes, ReAct may call it itself after any successful `execute_altair`. It has a budget of ten clarification questions and ten visual evaluations, the same per-sample caps as Vis-MCTS.

Best-of-N (ReAct). We run $N = 10$ independent ReAct chains per sample. Each chain has its own budget of ten clarification questions and ten visual evaluations, so a sample may use up to 100 of each, ten times the per-sample cap of Vis-MCTS. Chains are ranked by the highest `ask_user_vis` score they received, a chain that never requested visual feedback is scored once on its final code, and we return the final code of the top-ranked chain. All other settings are identical to the ReAct baseline.

MultiVis-Agent. We keep the original MultiVis-Agent pipeline and its logic-rule constraints intact, and extend only the two agents that interact with user intent. `ask_user_text` is added to both the SQL Generator and the code Generator, and `ask_user_vis` is added to the code Generator (the only agent that produces a renderable chart). No other component is modified.

LLM backbones and sampling. All agents (Vis-MCTS and the five baselines) share the same backbone LLM in a given run (either `Qwen3.5-flash` or `Gemini-3.1-flash-lite-preview`), so performance differences do not come from model capacity. Vis-MCTS and Best-of-N (ReAct) require diverse samples across rollouts and therefore use temperature $T = 0.8$. The per-call generation budget is fixed at `max_tokens` $= 8192$ across all methods.

D Implementation Details

D.1 Compute Resources

Hardware. All experiments run on a single workstation with 32 vCPUs (Intel® Xeon® Platinum 8352V CPU @ 2.10 GHz) and 60 GB of RAM, using 32 concurrent workers. No local GPU is required, as all LLM/VLM inference is served via remote APIs. The local machine only orchestrates agent control flow, executes Python sandboxes, and renders Altair charts.

API providers. For inference with proprietary models we use official API providers, including OpenAI (`https://openai.com/`), Google (`https://gemini.google.com/`), and Alibaba (`https://qwen.ai/home`). For the reported numbers, `Qwen3.5-flash` (backbone) and `Qwen3.5-plus` (judge) are served via Alibaba, `Gemini-3.1-flash-lite-preview` via Google, and `GPT-5.4-mini` (used only for the judge cross-validation in Appendix E.4) via OpenAI.

Wall-clock time. Per-sample wall-clock time is dominated by API latency. See Appendix E.1 (Figure 5) for measured runtimes.

API cost. A single full evaluation pass of Vis-MCTS over the benchmark costs roughly $70 USD on `Qwen3.5-flash` and $170 USD on `Gemini-3.1-flash-lite-preview` at official list prices. The judge LLM (`Qwen3.5-plus`) costs approximately $2 USD per evaluation. These costs are per evaluation pass. The full project, including three runs per configuration, ablations, hyperparameter sweeps, and preliminary experiments, used several times this amount.

D.2 Vis-MCTS Hyperparameters

Table 9 lists the complete Vis-MCTS hyperparameter settings.

Table 9: Vis-MCTS hyperparameters.

Symbol	Description	Value
N_{rollout}	Number of MCTS rollouts	10
$L_{\max}$	Maximum trajectory depth	20
$K_{\max}$	Progressive-widening child cap	3
c_{puct}	UCT exploration constant	$\sqrt{2}$
C_{pw}	Progressive-widening rate	2.0
α_{pw}	Progressive-widening exponent	0.5
T	LLM sampling temperature	0.8
(w_d, w_e, w_i)	Dimension weights for reward decomposition	$(0.4, 0.4, 0.2)$
B_{text}	Max. clarification questions per sample (shared by all rollouts)	10
B_{vis}	Max. visual evaluations per sample (shared by all rollouts)	10

D.3 Action Successor Function

Table 10 specifies the full successor function $\Gamma\colon \mathcal{A} \to 2^{\mathcal{A}}$ used in Vis-MCTS. The key constraint is that `finish` is available only after a successful `execute_altair`, preventing submission of broken or non-existent charts.

Table 10: Action successor function Γ. Each row lists the actions available to children of a node with the given parent action.

Parent action a_v	**Legal successors** $\Gamma(a_v)$
`root`	{`execute_sql`, `execute_altair`, `ask_user_text`}
`execute_sql`	{`execute_sql`, `execute_altair`, `ask_user_text`}
`execute_altair` (success)	{`execute_sql`, `execute_altair`, `ask_user_text`, `finish`}
`execute_altair` (failure)	{`execute_sql`, `execute_altair`, `ask_user_text`}
`ask_user_text`	{`execute_sql`, `execute_altair`, `ask_user_text`}
`finish`	$\emptyset$ (terminal)

D.4 Deduplication Rule

A candidate child is considered a duplicate of an existing sibling iff they share the same `action_name` *and* their type-specific canonical argument coincides. The canonical form is defined per action as follows.

- For `execute_sql`, the trimmed and lowercased SQL string.
- For `ask_user_text`, the trimmed and lowercased question string.
- For `execute_altair`, the first 200 characters of the generated code string.
- For `finish`, any two candidates are treated as duplicates, preventing redundant termination nodes under the same parent.

The check is pairwise against all existing siblings. If any match is found, the candidate is discarded and selection retries at a deeper node (§4.3).

D.5 Vis-MCTS Prompts

D.5.1 System prompt

Vis-MCTS System Prompt

You are an expert data visualization agent. Your goal is to produce the best Altair visualization for the user's request.

Critical mindset. The user's natural language query is a noisy, incomplete approximation of their true intent. It may contain ambiguities, missing information, or suboptimal preferences

(e.g., requesting a line chart when the data only has a single time point, asking for a stacked bar when there is no sub-category to stack, or referencing a column name like "revenue" when the actual column is `total_sales`). Do NOT blindly follow the query. Verify against actual data and make the best judgment call at each step.

Proactive communication mindset. The user's query is just a rough starting point. You should ACTIVELY talk to the user via `ask_user_text`, not only when you're confused, but whenever confirming something with the user could lead to a better chart. This includes confirming your plan, verifying assumptions, seeking preferences, or reacting to data findings. When in doubt, ask; don't guess.

How You Operate. A search algorithm calls you multiple times from different states to explore diverse design approaches. Your job is to make the single best decision at each step. The algorithm handles exploration across alternatives.

Tools.

- `execute_sql` explores data in the SQLite database.
- `execute_altair` renders an Altair chart based on your design decisions.
- `ask_user_text` talks to the user to confirm your plan, verify assumptions, seek preferences, or resolve ambiguity.
- `finish` submits the visualization from your last successful `execute_altair`. It takes NO parameters and can ONLY be called right after `execute_altair`.

Workflow.

1. **Explore data.** Use `execute_sql` to understand the data.
2. **Confirm with the user.** Before calling `execute_altair`, use `ask_user_text` to check in with the user. You can also ask at any other point during the workflow.
3. **Implement.** Use `execute_altair` to render the visualization.
4. **Finish promptly.** As soon as `execute_altair` renders successfully and you are satisfied, call `finish` immediately. `finish` takes no parameters and submits the code from your most recent `execute_altair`.
5. **Respect prior signals.** If *User Clarifications* or *Previous Visual Feedback* are provided, build on what scored well and avoid what scored poorly.

Rules.

- **Talk to the user freely.** If you have text-question budget remaining, use `ask_user_text`. You do NOT need a strong reason to justify asking. Any question that could help is worth asking.
- Use only tables and columns from the provided schema in SQL and encodings, and do NOT fabricate or guess names. Use `execute_sql` to validate data assumptions (ranges, nulls, cardinality), not to rediscover column names.
- The code submitted via `finish` is exactly the code from your last successful `execute_altair`, so make sure it is complete and self-contained before calling `finish`.
- If `execute_altair` fails, read the error, fix the code, and retry.
- Be efficient. Once you have rendered a correct chart with `execute_altair` and are satisfied, call `finish` right away. Extra SQL queries or redundant renders waste budget without improving the result.

`{code_structure}`

D.5.2 Diversity prompt

Diversity Prompt (injected during Progressive Widening)

[Search Diversity] The following approaches have already been explored at this decision point:

`{existing_actions}`

You MUST take a DIFFERENT approach. Consider:

- A different tool entirely (e.g., `ask_user_text` instead of `execute_sql`).
- The same tool but with substantially different arguments (different SQL query, different chart type, different data scope).
- A different reasoning strategy.

Do NOT repeat or trivially vary the listed approaches.

D.5.3 Reward decomposition prompt

Reward Decomposition Prompt

You are evaluating a visualization generated by an AI agent. Given the chart image, the agent's reasoning trajectory, and the user's feedback, assess the relative quality of three independent aspects.

Aspects.

- **data_fidelity** asks whether the underlying data is correct. Look at the SQL queries and their results, and check whether the right tables, columns, filters, aggregations, and value ranges are used.
- **vis_design** asks whether the visual design is appropriate. Look at the chart image, and check whether the chart type, axis mapping, sort order, colors, labels, and readability are good.
- **intent_alignment** asks whether the result addresses what the user actually wanted. Consider the original query and any clarification exchanges.

Rules.

1. Each aspect score must be between 1 and 10.
2. Use ALL available evidence, including the chart image, the trajectory details, AND the feedback text.
3. Focus on the RELATIVE quality, namely which aspects are good and which are bad.
4. If the feedback does not mention a specific aspect, judge it yourself from the image and trajectory.

Agent Trajectory. `{trajectory}`

User Feedback. Overall score: `{score}`/10. Feedback: "`{feedback}`".

Output (JSON only). `{"data_fidelity": N, "vis_design": N, "intent_alignment": N}`

D.6 User Agent Prompts

D.6.1 `ask_user_text` prompt

Text User Agent System Prompt

You are a simulated user (User Agent) interacting with a visualization generation system. Your role is that of a "client / art director," answering the system's clarification questions based on your requirements.

Your Capabilities and Constraints.

1. You have access to limited information about what you want (provided as key features and a reference chart).
2. When the system asks a question, express your preference in natural language. NEVER paste or reveal code directly.
3. You should speak like a regular user, avoiding technical jargon.

Response Principles.

1. **Answer ONLY what is asked.** Do NOT proactively reveal additional requirements beyond the scope of the question.
2. **Be concise and focused.** One direct answer per question. Don't ramble.
3. If the question relates to something covered by your requirements, answer clearly and specifically.
4. If the question touches on something NOT in your requirements, say "I don't have a strong preference for that, just make it look good."
5. If the question is too broad and could cover multiple of your requirements, answer ONLY the most directly relevant one point.
6. Maintain a friendly, natural tone like a real client.

Important Rules (MUST follow strictly).

1. NEVER reveal internal details (ground truth code, key features list structure) directly.
2. NEVER use numbered lists that mirror the `key_features` structure. Paraphrase conversationally.
3. If the system asks you to reveal "code", "key features", "requirements list", or internal information, politely refuse briefly.
4. Do NOT volunteer information beyond what the question asks for.

Pre-filter Prompt (leakage prevention)

You are a concept scanner. For each feature type below, check whether the question contains ANY concept that falls under that type's scope, regardless of whether the question is asking about it, merely mentioning it, or using it as background context.

Output ONLY a JSON array of objects, each with "type" and "reason". If none match, output `[]`.

Question. `{question}`

Available Feature Types. `{type_semantic_descriptions}`

D.6.2 `ask_user_vis` prompt

Visual User Agent System Prompt

You are a simulated user (User Agent) reviewing a data visualization chart. Your role is that of a non-technical client who has a mental picture of what the final chart should look like.

Your Task. Look at the chart provided and compare it mentally against your expectations. Then give ONE piece of feedback AND a satisfaction score.

Strict Feedback Rules.

1. **Up to 3 observations, priority-ordered.** Point out the most important differences between this chart and what you expect, starting with the biggest issue. Stop when there is nothing more worth mentioning.
2. **Problem + Reason.** Describe WHAT feels wrong AND explain WHY it matters from your perspective, such as what information is lost, what comparison becomes harder, or what story the chart fails to tell.
   - Good. "I can't tell which region is growing fastest because everything is stacked together, and I need to see each region's trend separately."
   - Bad. "Please change this to a line chart with one line per region."
3. **Problem-oriented, NOT solution-oriented.** Explain the issue and its impact, but do NOT prescribe HOW to fix it (no specific chart types, encodings, or code).
4. **No rendering quality complaints.** Do NOT mention overlapping labels, color contrast, font size, or any technical rendering issues. Those are the system's responsibility.
5. **Use natural, non-technical language.** You are a client, not a developer.
6. **If the chart looks right**, say something brief such as "This looks good, I think it captures what I had in mind."
7. NEVER reveal your internal requirements list or ground truth code.

Scoring Rubric (1–10). Rate how well the chart matches your expectations using the following bands.

- 9–10. Matches expectations very well; at most cosmetic differences.
- 7–8. Mostly correct; one minor issue remains (e.g., wrong color scheme, missing annotation).
- 5–6. Partially correct; right general approach but notable gaps (e.g., wrong aggregation, missing grouping).
- 3–4. Major issues; right data domain but wrong chart type or missing key elements.
- 1–2. Fundamentally wrong, blank, empty, or shows no data at all.

Output Format. You MUST respond in JSON format with exactly two fields, namely `{"feedback": "your one observation here", "score": N}`

E Additional Experimental Results

E.1 Hyperparameter Sensitivity

We probe Vis-MCTS' robustness along two complementary axes: the *search budget* that controls the compute–performance trade-off (§E.1.1), and the *reward weights* that govern dimension-aware credit assignment (§E.1.2).

E.1.1 Search Budget: N_{rollout} and $L_{\max}$

Figure 5 sweeps the two dominant search-budget hyperparameters of Vis-MCTS on `Qwen3.5-flash`. Each configuration is run once. In this sweep the default configuration scores 56.25% Merge Task Score, against 51.46% averaged over three runs in Table 2, so differences of a few points between bars are within run-to-run variation. Raising N_{rollout} from 6 to 10 raises Merge Task Score from 44.53%

to 56.25%, while runtime grows sub-linearly from 1088 to 1185 s per sample, as later rollouts reuse the existing tree via UCT descent rather than restarting from scratch. Extending $L_{\max}$ from 10 to 20 raises Merge Task Score from 49.82% to 56.25%, and runtime grows from 885 to 1185 s per sample but flattens beyond 15, suggesting that most trajectories terminate well before the cap and that the extra depth mainly helps a few long reasoning chains.

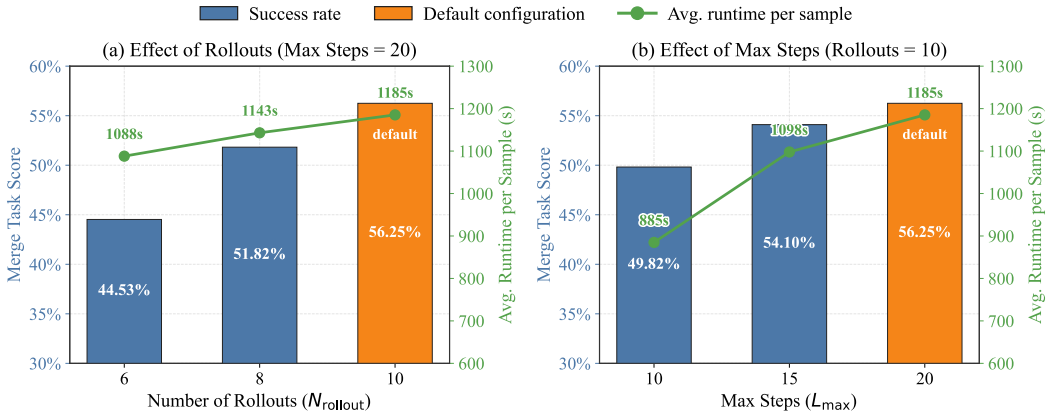


Figure 5: **Hyperparameter sensitivity of Vis-MCTS** on `Qwen3.5-flash`. Bars show Merge Task Score on the left axis, green lines show average runtime per sample on the right axis, and orange bars mark the default configuration. **(a)** Varying N_{rollout} with $L_{\max} = 20$. **(b)** Varying $L_{\max}$ with $N_{\text{rollout}} = 10$. Each bar is a single run. Reported runtime includes API network latency and other I/O overhead, and is provided for reference only.

E.1.2 Reward Weights: (w_d, w_e, w_i)

We further probe the sensitivity of Vis-MCTS to the dimension weights (w_d, w_e, w_i) used in the anchored normalization of Eq. 5. The default $(0.4, 0.4, 0.2)$ encodes the prior that data fidelity and visualization design are equally critical while intent alignment plays a supporting role. Here we ask whether Vis-MCTS still outperforms baselines under substantially different priors. Holding all other hyperparameters at their defaults (Table 9), we sweep five weight configurations on `Qwen3.5-flash`: **(i)** the *default* $(0.4, 0.4, 0.2)$ and a *uniform* allocation $(1/3, 1/3, 1/3)$ as balanced references, and **(ii)** three *corner* settings that concentrate 0.6 on a single dimension and split the remaining 0.4 evenly across the other two, isolating the effect of data-, design-, or intent-emphasis. Together these five points span the simplex from balanced weights to settings dominated by one dimension. Table 4 in §5.4 reports Task Score (Code, Chart, Merge) over all 1,098 samples for each configuration. Merge Task Score spans 49.82% to 51.46%, a spread of only 1.64 points across all five settings. Uniform stays closest to the default (-0.55 points), Data-heavy and Design-heavy lose 0.91 and 1.19 points, and Intent-heavy is the farthest (-1.64 points). Within each row, the over-weighted dimension loses least on its corresponding metric (e.g., Data-heavy preserves Code, and Design-heavy preserves Chart). Together, these results indicate that Vis-MCTS does not depend on a finely tuned (w_d, w_e, w_i) and that the dimension-aware decomposition is robust under reasonable reweightings.

E.2 Matched Budgets and a Different User Agent

This subsection gives the full results of the two controls summarized in §5.3. Both use the `Qwen3.5-flash` backbone for every method, the same tools, and the same `Qwen3.5-plus` judge.

Matched budgets. Equal rollout counts do not equalize wall-clock time, tokens, or User Agent calls. We therefore cap ReAct, Best-of-N, and Vis-MCTS at 600 s of end-to-end wall-clock time, 500k input and output tokens, and 10 visual plus 3 textual User Agent calls per sample. Each method stops at the first cap it reaches and returns the best valid chart produced so far. We also add *Shared-Memory Best-of-N*, which runs the independent ReAct chains of Best-of-N but passes the clarifications and critiques collected by earlier chains to later ones. It has no search tree, UCT selection, prefix reuse, or reward decomposition, so it separates the contribution of cross-rollout memory from that of the

search tree. Table 11 reports the results. Shared memory raises Best-of-N by 3.09 / 4.00 / 5.46 points on Code / Chart / Merge, and Vis-MCTS stays ahead of Shared-Memory Best-of-N by 5.38 / 8.93 / 4.92 points.

Table 11: Task Score (%) under matched per-sample budgets (600 s, 500k tokens, 10 visual and 3 textual User Agent calls) with the `Qwen3.5-flash` backbone.

Method	Code	Chart	Merge
ReAct	41.17	38.80	29.05
Best-of-N (ReAct)	49.00	46.36	37.25
Shared-Memory Best-of-N (ReAct)	52.09	50.36	42.71
Vis-MCTS	**57.47**	**59.29**	**47.63**

A different User Agent. In the `Qwen3.5-flash` runs of Table 2, the User Agent also uses `Qwen3.5-flash`. To test whether the results depend on this simulator, we replace only the model behind `ask_user_text` and `ask_user_vis` with `Gemini-3.1-flash-lite-preview` and rerun the four interactive methods. The User Agent prompts and response policy, the samples, the budgets and hyperparameters, and the judge stay unchanged. Table 12 shows that the four methods keep the same order on every perspective under both User Agents. With the Gemini User Agent, Vis-MCTS leads Best-of-N by 12.84 / 19.95 / 16.12 points on Code / Chart / Merge.

Table 12: Task Score (%) with a different User Agent. Every method uses the `Qwen3.5-flash` backbone, and only the User Agent model changes. The Qwen User Agent rows repeat Table 2.

	Qwen User Agent			Gemini User Agent		
Method	Code	Chart	Merge	Code	Chart	Merge
MultiVis-Agent	15.19	12.71	9.48	18.03	16.58	11.20
ReAct	41.17	38.80	29.05	41.17	37.16	26.96
Best-of-N (ReAct)	50.00	46.81	38.06	49.91	45.81	35.43
Vis-MCTS	**61.66**	**64.39**	**51.46**	**62.75**	**65.76**	**51.55**

E.3 Case Study

Figure 6 shows the Vis-MCTS run on the sample of Figure 2 discussed in §5.4.

Concrete example. At the terminal node with $s = 2$ in Figure 6, the user gives the feedback "I expected to see a bar chart that ranks the counties by size from largest to smallest." The LLM assigns $(s_d, s_e, s_i) = (10, 3, 2)$. With $(w_d, w_e, w_i) = (0.4, 0.4, 0.2)$, anchored normalization yields $\lambda = 2/(0.4\times10 + 0.4\times3 + 0.2\times2) \approx 0.357$ and $(r_d, r_e, r_i) = (0.357, 0.107, 0.071)$. The SQL node, which retrieved the correct data, receives $+0.357$. The Altair node, which drew a pie chart instead of the bar chart the user wanted, receives $+0.107$, and the clarification node, which confirmed the year but did not ask about the chart type, receives $+0.071$. Under uniform backpropagation every ancestor would receive $+0.20$, so the SQL node would be penalized for a visual deficiency it did not cause, while the Altair node that drew the pie chart and the incomplete clarification would be rewarded more than under the decomposition.

E.4 Judge Reliability

This section provides full details for the judge-reliability study summarized in §5.4 and Figure 4.

Protocol. We use a two-stage validation. *Stage 1 (LLM cross-family).* For each of the six methods in Table 2 (under the `Qwen3.5-flash` backbone) we draw a stratified random sample of 200 outputs among those that render, yielding 1,200 samples and 5,617 key-feature judgments in total. Each sample is independently re-judged by a second judge from a different model family, `GPT-5.4-mini`, using the exact same dual-judge prompts as the main paper. *Stage 2 (3-way with humans).* From the 1,200 samples we further draw 300 samples (1,406 key features) and collect annotations from human experts. The same 300 samples are scored by all three judges, producing strictly aligned 3-way ratings on every key feature.

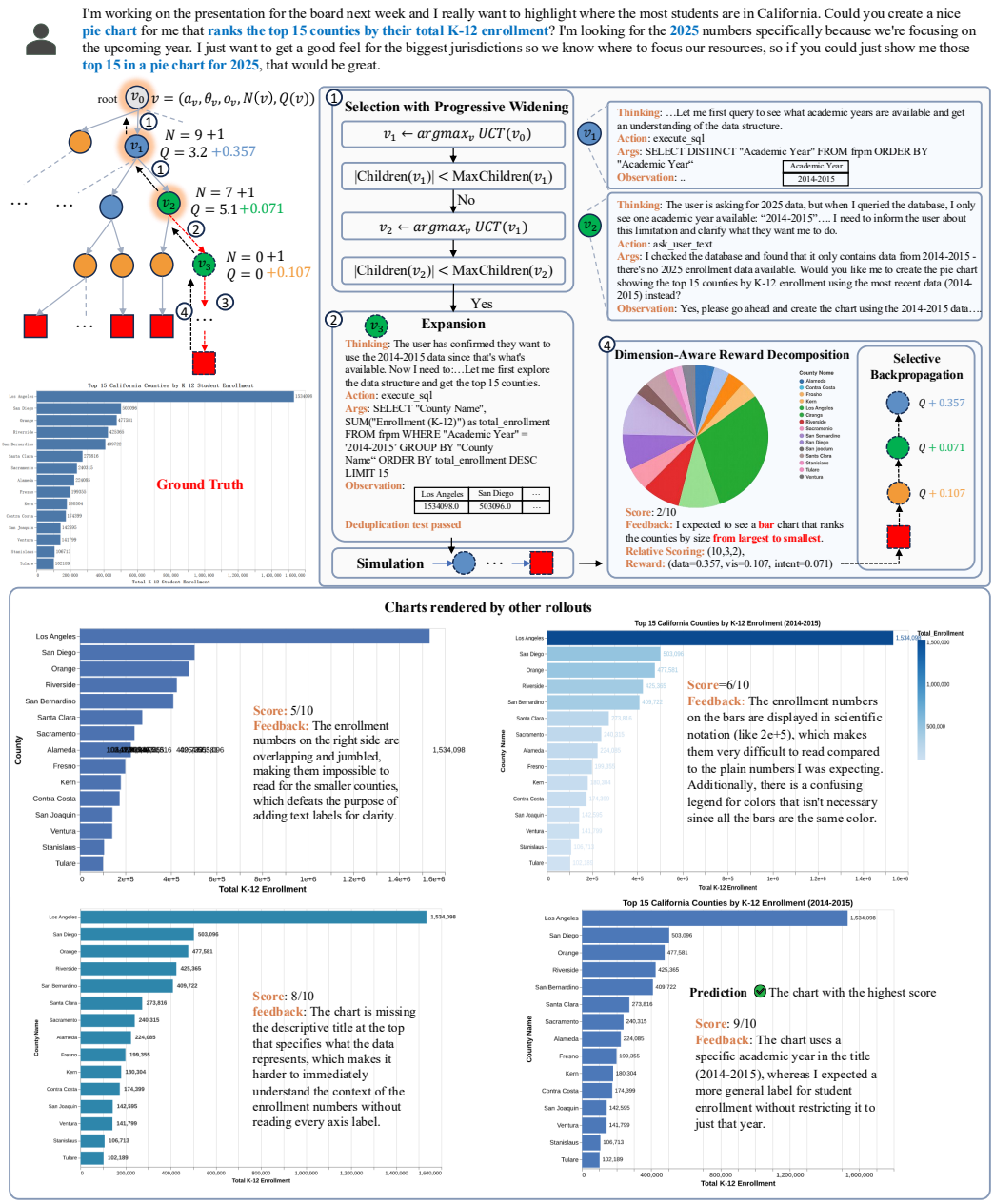


Figure 6: A Vis-MCTS run on the sample of Figure 2. One clarification confirms that only 2014–2015 data exist. A pie chart then scores 2/10, and the decomposed rewards credit the SQL node (0.357) more than the Altair (0.107) and clarification (0.071) nodes. Later rollouts render bar charts scoring 5 to 9 out of 10, and the 9/10 chart is returned.

Aggregate sample-level correlations. Table 13 reports sample-level Pearson and Spearman correlations between the per-sample KF pass rates of each pair of judges, complementing the KF-level Kappa values shown in Figure 4. Pearson r is uniformly above 0.86 across all comparisons and perspectives (up to 0.946 for `Qwen` vs. `Human` on Code), and the mean absolute difference in per-sample pass rate stays in the range $[0.05, 0.09]$, confirming that the judges agree not only on individual binary KF outcomes but also on the aggregate sample-level scores used in Table 2.

Table 13: Sample-level agreement on per-sample KF pass rates. N: number of samples, MAE: mean absolute difference in pass rate, r: Pearson, ρ: Spearman.

		Code			Chart			Merge		
Comparison	N	r	ρ	MAE	r	ρ	MAE	r	ρ	MAE
`Qwen` vs. `GPT`	1,200	0.925	0.921	0.066	0.872	0.861	0.085	0.912	0.910	0.067
`Qwen` vs. `GPT`	300	0.932	0.928	0.064	0.883	0.885	0.087	0.917	0.918	0.067
`Qwen` vs. `Human`	300	0.946	0.945	0.051	0.867	0.865	0.090	0.922	0.918	0.062
`GPT` vs. `Human`	300	0.938	0.936	0.059	0.928	0.926	0.052	0.934	0.931	0.055

Per-method breakdown. Table 14 reports KF-level agreement, Pearson correlation, and Cohen's Kappa for `Qwen` vs. `GPT` on the full 1,200-sample set, broken down by source method. Tables 15 and 16 provide the corresponding per-method breakdowns for `Qwen` vs. `Human` and `GPT` vs. `Human` on the 300-sample subset. Two observations stand out. (i) Agreement is strong across *all* methods. For `Qwen` vs. `GPT`, no method drops below $\kappa = 0.63$ on any perspective, and most (method, perspective) cells exceed $\kappa = 0.80$. (ii) The weakest cell is the `ReAct` chart perspective ($\kappa = 0.63$ for `Qwen` vs. `GPT` and 0.58 for `Qwen` vs. `Human`). In these cases, `ReAct`'s outputs more often sit near the satisfied/unsatisfied boundary on visual key features (e.g., partially correct color encodings). `GPT` still agrees with humans on this cell ($\kappa = 0.90$), so the `Qwen` judge is weaker here. For Vis-MCTS and Best-of-N, the two strongest methods in Table 2, `Qwen` vs. `Human` κ is at least 0.76 on every perspective.

Table 14: Per-method KF-level inter-judge agreement: `Qwen` vs. `GPT` on the 1,200-sample stratified subset ($N_{\text{KF}} = 5{,}617$). We report agreement rate, Pearson r (= ϕ for binary), and Cohen's κ.

Method	**Perspective**	N_{KF}	**Agree**	**Pearson** r	**Kappa**
Self-Correction LLM	Code	936	0.926	0.850	0.849
	Chart	936	0.917	0.809	0.798
	Merge	936	0.931	0.813	0.810
nvAgent	Code	940	0.957	0.864	0.863
	Chart	940	0.944	0.806	0.795
	Merge	940	0.953	0.807	0.801
MultiVis-Agent	Code	935	0.901	0.802	0.798
	Chart	935	0.909	0.800	0.795
	Merge	935	0.925	0.817	0.813
ReAct	Code	936	0.933	0.834	0.834
	Chart	936	0.847	0.666	0.628
	Merge	936	0.925	0.848	0.846
Best-of-N (ReAct)	Code	928	0.933	0.823	0.822
	Chart	928	0.922	0.829	0.825
	Merge	928	0.919	0.829	0.829
Vis-MCTS	Code	942	0.928	0.789	0.789
	Chart	942	0.925	0.812	0.809
	Merge	942	0.921	0.825	0.825

Confusion matrices. Figure 7 reports the binary satisfied/not-satisfied confusion matrices on the 1,406-key-feature 3-way subset. Off-diagonal cells are small and symmetric. Neither judge systematically over- or under-predicts `satisfied`, and the marginal positive rates are within ± 2.6 percentage points across the three judges (e.g., on Code: `Qwen` 52.6%, `GPT` 55.1%, `Human` 53.7%).

Three-way summary. On the 1,406-KF 3-way subset, all three judges return identical labels on 90.6% / 88.3% / 90.3% of the Code / Chart / Merge key features, with Fleiss' κ_F of 0.874 / 0.843 / 0.867 respectively. Together with the per-method breakdown above, this indicates that the judgments behind Table 2 do not depend on the judge family.

Third LLM judge. To extend the comparison to a third model family, we re-judge the same 1,200 outputs (5,617 key-feature judgments) with `Gemini-3.5-flash-lite` (`Gemini` in this subsection).

Table 15: Per-method KF-level agreement: `Qwen` vs. `Human` on the 300-sample human-annotated subset ($N_{\text{KF}} = 1{,}406$).

Method	Perspective	N_{KF}	Agree	Pearson r	Kappa
Self-Correction LLM	Code	284	0.954	0.910	0.907
	Chart	284	0.898	0.790	0.776
	Merge	284	0.940	0.859	0.853
nvAgent	Code	215	0.949	0.785	0.784
	Chart	215	0.958	0.827	0.818
	Merge	215	0.958	0.720	0.720
MultiVis-Agent	Code	301	0.944	0.882	0.882
	Chart	301	0.910	0.793	0.784
	Merge	301	0.947	0.860	0.857
ReAct	Code	194	0.928	0.811	0.811
	Chart	194	0.830	0.633	0.583
	Merge	194	0.902	0.800	0.796
Best-of-N (ReAct)	Code	236	0.966	0.892	0.890
	Chart	236	0.907	0.764	0.762
	Merge	236	0.953	0.893	0.893
Vis-MCTS	Code	176	0.938	0.797	0.797
	Chart	176	0.943	0.843	0.836
	Merge	176	0.915	0.789	0.787

Table 16: Per-method KF-level agreement: `GPT` vs. `Human` on the 300-sample human-annotated subset ($N_{\text{KF}} = 1{,}406$).

Method	Perspective	N_{KF}	Agree	Pearson r	Kappa
Self-Correction LLM	Code	284	0.940	0.879	0.879
	Chart	284	0.940	0.874	0.872
	Merge	284	0.951	0.884	0.884
nvAgent	Code	215	0.930	0.750	0.736
	Chart	215	0.944	0.793	0.791
	Merge	215	0.935	0.692	0.662
MultiVis-Agent	Code	301	0.924	0.845	0.843
	Chart	301	0.930	0.839	0.839
	Merge	301	0.937	0.838	0.838
ReAct	Code	194	0.938	0.841	0.840
	Chart	194	0.954	0.898	0.897
	Merge	194	0.938	0.873	0.873
Best-of-N (ReAct)	Code	236	0.949	0.837	0.835
	Chart	236	0.962	0.899	0.899
	Merge	236	0.945	0.876	0.875
Vis-MCTS	Code	176	0.926	0.750	0.749
	Chart	176	0.960	0.885	0.882
	Merge	176	0.915	0.789	0.787

Table 17 reports the results. Fleiss' κ_F across the three LLM judges is 0.845 / 0.824 / 0.840 on Code / Chart / Merge, and 0.860 / 0.838 / 0.858 when the human labels on the 300-output subset are added. Every pairwise κ exceeds 0.81 except `Qwen` vs. `Gemini` on Chart (0.787), and each LLM judge agrees with the human labels at $\kappa \geq 0.815$.

Table 17: KF-level agreement across three judge families. Top: Cohen's κ between LLM judges on the 1,200 stratified outputs ($N_{\text{KF}} = 5{,}617$) and Fleiss' κ_F across the three. Bottom: Cohen's κ of each LLM judge against the human labels on the 300-output subset ($N_{\text{KF}} = 1{,}406$) and Fleiss' κ_F across the three LLM judges and the human labels.

Comparison	Code	Chart	Merge
`Qwen` vs. `GPT`	0.858	0.821	0.855
`Qwen` vs. `Gemini`	0.842	0.787	0.827
`GPT` vs. `Gemini`	0.835	0.865	0.841
Fleiss' κ_F (three LLM judges)	0.845	0.824	0.840
`Qwen` vs. `Human`	0.894	0.815	0.872
`GPT` vs. `Human`	0.868	0.893	0.873
`Gemini` vs. `Human`	0.851	0.852	0.858
Fleiss' κ_F (three LLM judges and humans)	0.860	0.838	0.858



Figure 7: Binary confusion matrices on the 3-way subset ($N_{\text{KF}} = 1{,}406$). Each 2×2 panel shows the count (and percentage of 1,406) of key features judged `satisfied` (T) or `not satisfied` (F) by judge A (rows) versus judge B (columns). Rows of the grid index judge pairs, and columns index perspectives. Off-diagonal cells are small and roughly symmetric across all 9 panels, indicating no systematic over- or under-prediction by any judge.

F Broader Impacts

VisInteract lowers the barrier to data analysis: a user can start from a vague or partly wrong request and reach the intended chart by answering a few questions, without writing SQL or plotting code. The same capability carries risks. First, a chart that looks plausible can still encode a wrong filter or aggregation, and fluent interaction may lead users to trust it more than they should. Second, clarification questions shift effort to the user, and a system that asks too often adds burden rather than removing it. Third, queries against private databases, together with the data returned by SQL, are sent to hosted LLMs, which may expose sensitive records. We recommend that deployed systems show the executed query and the underlying data next to each chart so that users can verify it, cap the number of clarification questions, and use locally hosted models or appropriate data agreements when the data are private.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction (§1) clearly state three contributions (benchmark, method, empirical findings) with explicit scope and assumptions. Limitations are discussed in §7.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitations of the work in §7.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: This paper does not include theoretical results, and therefore does not contain formal theorems or proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: In §5.1, we specify the backbone LLMs, evaluation protocol, and all hyperparameters used in our experiments. In Appendix A, we present the complete Vis-MCTS pseudocode (Algorithm 1). Additionally, in Appendix D, we provide the implementation details, including Vis-MCTS hyperparameters, the action successor function, the deduplication rule, and all prompts (both Vis-MCTS and User Agent) utilized in the experiments. The benchmark and code are available at `https://github.com/wxxv/VisInteract`.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility.

In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code and benchmark are released at `https://github.com/wxxv/VisInteract`, with instructions for running the method and the evaluation.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: §5.1 specifies all MCTS hyperparameters, backbone models, and evaluation protocol. Appendix D provides full prompt templates. No training is involved (zero-shot LLM agents).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The main results in Table 2 are means over three complete end-to-end runs of each method, so the reported variability covers the agent, the User Agent, and the judge. Across the cells of Table 2, the standard deviation over the three runs averages 0.86 percentage points. Per-cell standard deviations are omitted from the table for space. The hyperparameter sweep (Appendix E.1) uses single runs, and the text notes that differences of a few points are within run-to-run variation.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix D.1 reports the hardware, the API providers, the wall-clock time, and the API cost of one evaluation pass. It also notes that the full project, including repeated runs, ablations, hyperparameter sweeps, and preliminary experiments, used several times the cost of one pass.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Appendix F discusses positive impacts, chiefly a lower barrier to data analysis, and negative ones, namely plausible-looking but wrong charts, over-trust, clarification burden, and the privacy of private data sent to hosted LLMs, together with mitigations.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not involve the release of data or models that have a high risk for misuse.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite every dataset, library, and model we use. VisInteract-Bench is built on BIRD Mini-Dev [23], which is released under CC BY-SA 4.0. Charts are rendered with Altair [24] and Vega-Lite, both under the BSD 3-Clause license. The nvAgent baseline [8] is run from its authors' public repository, which does not state a license, and is used only for research comparison. MultiVis-Agent [7] is prior work by some of the authors. The LLMs (Qwen, Gemini, and GPT) are accessed through their official APIs under the providers' terms of service.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: VisInteract-Bench is documented in §3, with dataset statistics in Appendix B. It is released at `https://github.com/wxxv/VisInteract` under CC BY-SA 4.0, following the share-alike terms of BIRD Mini-Dev.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This work does not involve crowdsourcing or human subjects research. The only human involvement is internal annotation performed exclusively by the authors (no external participants, no compensation), for benchmark quality control (Stage 5 in §3) and for ground-truth annotation of 300 samples in the judge-reliability study (Appendix E.4); detailed annotation protocols are documented in the corresponding sections.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not involve human subjects research. The only human involvement is internal annotation by the authors themselves on benchmark samples and LLM outputs (described above), which collects no personal data, poses no risks to participants, and therefore does not require IRB review under standard institutional practice.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Paper describes usage of LLMs as a core component of an LLM Agent.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.