

## Technical Appendices and Supplementary Material

- A: Detailed Related Work
- B: More Details of Synthetic Pipeline
- C: More Details of nvBench 2.0
- D: More Details of Experimental Setups
- E: More Details of Error Analysis
- F: Limitations
- G: Ethic Statement

## A Detailed Related Work

### A.1 *Text2VIS* Benchmarks

*Text2VIS* benchmarks play a crucial role in evaluating the performance of *Text2VIS* systems [45]. As the predecessor of nvBench 2.0, nvBench 1.0 [16] is a commonly used *Text2VIS* benchmark, constructs datasets by leveraging the semantic alignment between SQL and visualization query language, which is a SQL-like specification that defines the visualization structure and details the data transformation processes. It employs template-based structures to systematically translate VQL into NL. This structured approach facilitates end-to-end model training by enhancing the clarity of both inputs and outputs [28, 46, 47, 6, 48, 49]. Building on nvBench 1.0 [16], Dial-NVBench [19] introduces multi-turn dialogues, allowing models to capture user intent through iterative interactions. VisEval [29] further refines nvBench by filtering out ambiguous, irrational, duplicated, and incorrect queries using a three-step selection process (rule-based, LLM-based, and human-based), and offers an automated evaluation framework covering validity, legality, and readability. However, all three benchmarks [16, 19, 29] remain focused on well-specified queries that map directly to a single correct visualization, without explicitly addressing ambiguity in user intent.

To explore ambiguous and under-specified query formulations, ChartGPT [30] extends nvBench by prompting GPT-3 to generate more abstract and natural utterances compared to the original ones. Similarly, while some other *Text2VIS* datasets include ambiguous queries [31, 18, 17], they do not explicitly define ambiguity types and provide a complete set of valid chart results. Beyond the realm of *Text2VIS*, ambiguity has also been explored in *Text2SQL* benchmarks, where studies have considered data selection and computation ambiguity [13, 50], but they do not address ambiguity in the visualization space. While some *Text2VIS* systems have attempted to address ambiguity by detecting it [10, 51] or inferring underspecified queries [52], they lack a benchmark for systematic evaluation.

To fill this gap, we propose nvBench 2.0, the first ambiguity-aware *Text2VIS* benchmark, which provides ambiguous user queries and supports one-to-many mappings with multiple valid visualizations. By doing so, it enables a more comprehensive evaluation of *Text2VIS* systems in real-world scenarios.

### A.2 LLMs for Data Synthesis

Recently, the use of LLMs for data synthesis or data augmentation has become increasingly prevalent. Many studies leverage LLM-generated data for training models [32–38], as well as for evaluating the performance of other trained models [39]. In the NLP domain, researchers have utilized LLMs to generate synthetic data for tasks like text classification [40–42]. These works showcase that LLM-generated data can enhance data diversity, thereby improving model generalization and robustness. Building on this, VL2NL [15] extends LLMs to *Text2VIS* domain, generating natural language descriptions (*e.g.*, L1 and L2 captions, and user commands) from Vega-Lite specifications. Similarly, the application of LLMs for tabular data or database-related tasks has gained attraction. Common approaches for generating *Text2SQL* or table question answering datasets often involve generating TEXT queries first, followed by SQL generation [13, 50]. ScienceBenchmark [43] takes a reverse approach by starting with seed SQL queries, then generating new SQL queries from the domain schema, and translating them into natural language queries using fine-tuned LLMs. We follow this reverse construction philosophy in developing nvBench 2.0. Specifically, we begin by extracting VQL from seed charts and then use LLMs to reverse engineer the corresponding text descriptions. The advantage of this approach is that VQL clearly defines each step and the ambiguity types involved, allowing us to better capture one-to-many (TEXT, VIS) pairs.

By leveraging LLMs to generate multi-step reasoning data, the performance of models on long-chain and complex reasoning tasks can be further improved. As demonstrated by Hunter et al. [44], process supervision via multi-step reasoning significantly enhances model reliability on tasks such as mathematical problem-solving. Similarly, Step-DPO [23] shows that generating step-wise reasoning data enables models to better capture intermediate steps, resulting in improved accuracy. Following this approach, we also generate multi-step reasoning data for tasks in the *Text2VIS* domain, where each step of the reasoning process is explicitly defined, contributing to more accurate and interpretable model predictions.

## B More Details of Synthetic Pipeline

### B.1 Step 1: Ambiguity-aware VIS Tree Synthesis

Our ambiguity-aware visualization tree synthesis forms the foundation for synthesizing ambiguous *Text2VIS* data. As shown in Figure 6, this process injects ambiguities into a seed visualization.

**Transforming the Seed Visualization into a Tree Abstraction.** Given a data table  $D$  and a seed visualization  $v$  (e.g., (Figure 6-①), we first convert the  $v$ —along with its underlying query—into an Abstract Syntax Tree (AST), which we refer to as the seed visualization tree  $T$  (e.g., Figure 6-②). The grammar of AST is based on the predecessor work, nvBench [16]. This tree explicitly encodes all design decisions made in creating  $v$  and is formally defined as:

$$v \mapsto T = \{\mathbf{A} \mid \mathbf{A} = [a_1, a_2, \dots, a_t]\} \quad (2)$$

Here, each node  $a_i$  represents a construction action for a visualization component as a tuple  $(\tau, op, params)$ , where:

- $\tau \in \{\text{explicit}, \text{ambiguous}, \text{implicit}\}$  denotes the ambiguity type of the action node;
- $op$  specifies the operation (e.g., data selection, chart type selection, channel mapping, data transformation selection, etc.);
- $params$  contains the specific parameters for the operations.

#### Controlled Ambiguity Injection.

We then transform  $T$  into an ambiguity-aware tree  $T'$  through three operations:

- **Injecting ambiguous nodes** : We add nodes that represent components with multiple valid interpretations. For example, replacing “Local Gross” with an ambiguous choice between “Local Gross” and “World Gross”.
- **Adding implicit nodes** : We include nodes for components not explicitly specified but required for visualization completion. For example, adding a node for the “COLOR” encoding channel.
- **Modifying explicit nodes** : We adjust certain explicit nodes to account for potential ambiguities. For example, changing a “Mark” node initially set as “Bar” into an ambiguous choice among various mark types or requiring inference from analytic tasks.

By applying these steps, the resulting ambiguity-aware tree  $T'$  captures the full range of possible interpretations for the seed visualization. For example, as shown in Figure 6-③, this tree contains some new nodes such as:

**A1** : (ambiguous, data\_column, {field:[Local\_Gross, World\_Gross]})

**A2** : (explicit, task, {value:[Trend]})

**A3** : (implicit, data\_value, {value:[Comedy, Action]})

#### Ambiguity Metadata Generation for Ambiguity Injection.

To enable precise ambiguity injection in data tables, we propose a systematic metadata generation process that integrates structured knowledge bases with large language models (LLMs). This process

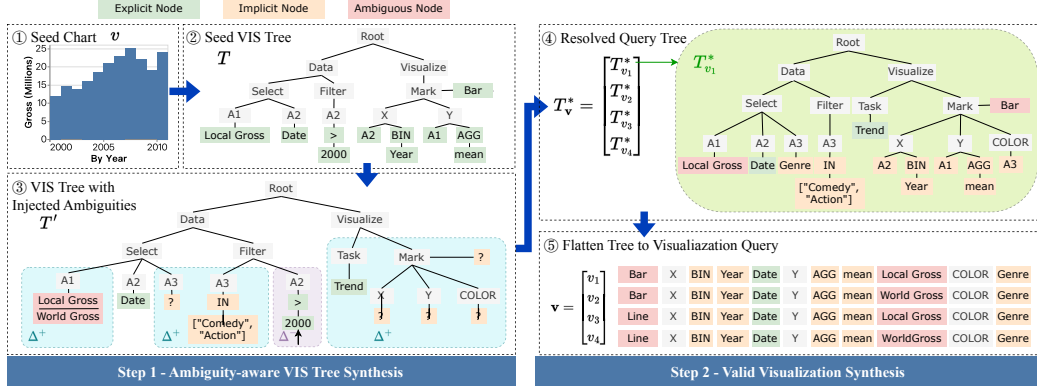


Figure 6: Injecting ambiguities into a seed visualization.

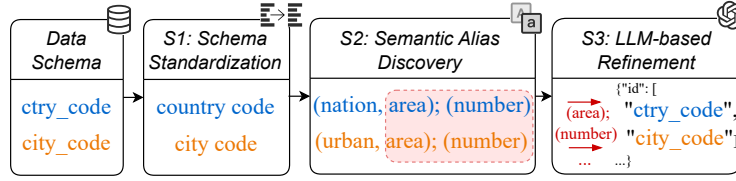


Figure 7: Ambiguity metadata generation workflow.

identifies and categorizes potential semantic ambiguities in table schemas, producing metadata that guides the construction of ambiguity-aware visualization trees. Each node in the visualization tree  $T'$  is labeled as ambiguous, implicit, or explicit based on the metadata, ensuring visualizations reflect multiple valid query interpretations. The process comprises three key stages: schema standardization, semantic alias discovery, and LLM-based refinement.

**Stage 1: Schema Standardization:** The first step involves standardizing the original data schema by refining or expanding column names. Abbreviated or domain-specific terms are transformed into more descriptive, conventional labels. For example, a column labeled `ctry_code` is standardized to `country code`. Such standardization forms a clearer basis for subsequent ambiguity analysis.

**Stage 2: Semantic Alias Discovery:** After standardizing the schema, we leverage ConceptNet [53] to identify potential semantic aliases for each column name. ConceptNet’s multilingual knowledge graph provides synonyms, hypernyms, and other semantically related terms, helping detect conceptual overlaps. We flag pairs of columns with similar meanings or concept overlap as potential sources of ambiguity. For example, `country code` and `city code` may both have meanings related to `number`, introducing possible confusion in user queries.

**Stage 3: LLM-Based Refinement:** We refine the flagged ambiguous column pairs using GPT-4o-mini with a chain-of-thought (CoT) prompting strategy. The model analyzes the original column names, their standardized forms (Stage 1), and the ConceptNet-derived aliases and ambiguity flags (Stage 2). It then generates a final, validated set of ambiguous pairs, which is formatted into a JSON metadata file. For example, as shown in Figure 7, one of the identified ambiguous pairs is `ctry_code` and `city_code` due to their similar word aliases. This process supports ambiguity-aware visualization generation and step-wise reasoning.

By combining these stages, we generate the necessary metadata to guide the construction of ambiguity-aware visualization trees, ensuring that each node is accurately marked as explicit, ambiguous, or implicit, thus enabling the synthesis of visualizations that reflect multiple valid interpretations of the query.

## B.2 Step 2: Valid Visualization Synthesis

Once we have an ambiguity-aware visualization tree  $T'$ , the next stage is to generate a set of valid visualizations  $\mathbf{v} = \{v_1, v_2, \dots, v_k\}$ . Each visualization  $v_i$  represents one possible resolution of the ambiguities present in  $T'$  (see Figure 6-3). In this step, we define a resolution function  $\mathcal{R}$

to systematically clarifies ambiguous and implicit nodes, transforming  $T'$  into a set of resolved trees  $\{T_{v_1}^*, \dots, T_{v_k}^*\}$  (see Figure 6-④). Each resolved tree  $T_{v_i}^*$  is then “flattened” into a concrete visualization query  $v_i$  (see Figure 6-⑤).

**Task Description.** Recap that a partially ambiguous visualization tree  $T'$  may contain:

- *Ambiguous nodes*: Multiple valid interpretations (*e.g.*, which column to use for “gross”).
- *Implicit nodes*: Necessary but unspecified details (*e.g.*, binning a date field by year).
- *Explicit nodes*: Directly specified components (*e.g.*, “bar” mark).

To produce valid visualizations, these ambiguous and implicit nodes must be resolved in a manner consistent with established visualization grammar rules (*e.g.*, requiring temporal fields to be binned). Formally, we define:

$$\mathcal{R}(T') \rightarrow \{T_{v_1}^*, T_{v_2}^*, \dots, T_{v_k}^*\} \quad (3)$$

where each  $T_{v_i}^*$  is a resolved tree that has no remaining ambiguity or unspecified details. The flattening process then converts each  $T_{v_i}^*$  into a finalized visualization specification  $v_i$ . This yields the complete set of valid visualizations:  $\mathbf{v} = \{v_1, v_2, \dots, v_k\}$ .

In the following sections, we describe how an Answer Set Programming (ASP) solver [22] is used to implement the resolution function  $\mathcal{R}$  while ensuring that each resolved visualization adheres to the necessary grammar constraints.

**ASP Solver Objective.** ASP is a declarative constraint programming paradigm well-suited for knowledge representation and reasoning [22, 54, 55]. Encoding the ambiguity resolution process and grammar rules as logical constraints has the following benefits:

- *Completeness*: The solver can enumerate all stable models (*i.e.*, all possible ways to resolve ambiguous or implicit nodes) that satisfy the visualization grammar.
- *Correctness*: Only solutions that meet mandatory constraints (*e.g.*, “temporal fields must be binned”) are considered valid.
- *Diversity*: Each output corresponds to a distinct interpretation of the query, ensuring coverage of all plausible visualizations.

The number of resulting visualizations,  $k = |\mathbf{v}|$ , represents the **ambiguity level**—how many distinct interpretations the solver deems valid for the given  $T'$ . After obtaining these solutions, we can filter or select a subset based on a target ambiguity level  $k$ , ensuring that each retained visualization differs from the others.

**ASP Syntax Overview.** ASP is built on a logical foundation with several key syntactic constructs [22]. The fundamental unit in ASP is a rule of the form: `Head :- Body.`, which states that the head is true if all literals in the body are satisfied. For example, the rule: `light_on :- power_available, switch_flipped.` expresses that the light will be on if both power is available and the switch is flipped.

Some special cases include:

- *Facts*: Rules without a body represent unconditional truths. For example, `power_available.` asserts that power is available.
- *Integrity Constraints*: Rules without a head prohibit certain combinations of conditions. For example, the constraint: `:- not power_available, light_on.` ensures that the light cannot be on when power is not available.

An ASP program consists of a collection of rules, facts, and constraints that collectively define a search space. The ASP solver then computes all stable models (*i.e.*, answer sets) that satisfy these conditions. Each stable model represents a valid system state or, in our context, a valid resolution of the ambiguous visualization tree.

For example, consider a simple lighting system modeled with:

- *Rule*: `light_on :- power_available, switch_flipped.`

1092 • *Fact:* `power_available.`, `switch_flipped.`

1093 Given these statements, the ASP solver determines the unique answer set containing `light_on`, as all  
1094 conditions in the rule body are satisfied. If we instead had `not switch_flipped.`, the solver would  
1095 exclude `light_on` from the answer set.

1096 By exhaustively computing all stable models that meet the specified constraints, the ASP solver  
1097 identifies all valid visualization configurations implied by our ambiguity-aware visualization tree. This  
1098 systematic resolution is key to generating a complete set of valid visualizations from an ambiguous  
1099 query.

1100 **ASP Rules for Resolving Ambiguity-aware Visualization Tree.** We formalize the visualization  
1101 design space using ASP by converting each node in the ambiguity-aware visualization tree  $T'$  into  
1102 ASP rules. As defined in Section B.1, each node in the visualization tree is represented as a tuple  
1103 (*type, operation, parameters*), which is mapped into ASP entities, (e.g., like `entity(E, _, _)`).  
1104 and their associated attributes (e.g., like `attribute(A, _, _)`).

1105 *Rules for Explicit Nodes.* Nodes that directly specify a visualization component are encoded as entities  
1106 with fully defined attributes. For example, a node indicating a specific mark selection, such as a bar  
1107 chart, is encoded in ASP as:

1108 • `entity(mark, parent_id, mark_id).`  
1109 • `attribute((mark, type), mark_id, bar).`

1110 These rules explicitly assert that the mark type is “bar”.

1111 *Rules for Ambiguous Nodes.* Nodes that allow multiple valid interpretations are encoded using ASP  
1112 choice rules. For example, if an encoding node can correspond to either “temp\_max” or “temp\_min”,  
1113 we encode this ambiguity as follows:

1114 • `1 { attribute((encoding, field), e_id, temp_max); attribute((encoding, field),`  
1115 `e_id, temp_min) }.` ensures at least one option should be selected.  
1116 • An accompanying integrity constraint ensures that only one of the two options is selected: `:- at`  
1117 `tribute((encoding, field), e_id, temp_max), attribute((encoding, field), e_id,`  
1118 `temp_min).`

1119 This formulation forces the solver to choose exactly one interpretation for each ambiguous node.

1120 *Rules for Implicit Nodes.* Implicit nodes represent necessary components that are not explicitly  
1121 specified in the query. These nodes are encoded using placeholder attributes to indicate that the value  
1122 is not determined. For example, a mark node with an unspecified chart type is represented as:

1123 • `entity(mark, parent_id, mark_id).`  
1124 • `attribute((mark, type), mark_id, _).`

1125 This indicates the mark exists, but its type is undetermined.

1126 To capture the complete visualization design space, we also encode comprehensive design knowledge  
1127 as ASP rules [56, 54, 55], which fall into three categories:

1128 Definition Rules for Visualization. Declarative statements that establish foundational visualization  
1129 elements, such as available chart types or encoding channels. For example, `domain((mark,`  
1130 `type), (point; bar; pie)).` defines that the mark type for a chart can be point, bar, or pie.

1131 Hard Constraints for Visualization. Mandatory conditions that any valid visualization must sat-  
1132 isfy. For example, the constraint `violation(no_encodings) :- entity(mark, _, M), not`  
1133 `entity(encoding, M, _).` ensures that every mark has at least one visual encoding channel.

1134 Choice Rules for Visualization. Rules that govern the selection among multiple options when construct-  
1135 ing a visualization. For example `0 { attribute((encoding, field), E, N): domain((field,`  
1136 `name), N) } 1 :- entity(encoding, _, E).` ensures that each encoding is associated with at most  
1137 one field.

1138 **Applying ASP Solver to Reason Valid Visualization.** By encoding the ambiguity-aware visualization  
 1139 tree structure and design principles as ASP rules, we create a powerful mechanism to resolve  
 1140 ambiguities. The ASP solver explores all possible resolutions for ambiguous nodes, ensuring that only  
 1141 solutions adhering to the visualization grammar constraints are accepted. This results in a diverse set  
 1142 of valid visualizations, with variations in chart type, encoding mappings, and data transformations,  
 1143 while staying true to the original ambiguous query.

### 1144 B.3 Step 3: Ambiguous Text Query Synthesis

1145 As shown in Figure 2 (c), this step runs in parallel with the valid visualization synthesis described in  
 1146 Section B.2. Building on the ambiguity-aware visualization tree  $T'$ , this step aims to synthesize a  
 1147 corresponding ambiguous natural language query  $q$ .

1148 **Task Description.** Given the input ambiguous visualization tree  $T' = \{\mathbf{A} \mid \mathbf{A} = [a_1, a_2, \dots, a_h]\}$ ,  
 1149 the corresponding natural language query  $q$  is generated using the mapping function  $\mathcal{M}$ :

$$Q = \mathcal{M}(T') = [\mathcal{M}(a_1), \mathcal{M}(a_2), \dots, \mathcal{M}(a_h)] \quad (4)$$

1150 where the tuple of each visualization construction action  $a_i$  in  $T'$  is mapped to a corresponding natural  
 1151 language expression  $\mathcal{M}(a_i)$ .

1152 For a given  $T'$ , its corresponding  $q$  must satisfy the following conditions to ensure correctness:

- 1153 • *Completeness:* Ensure that all actions in the original  $T'$  are covered in the generated  $q$ :

$$\forall a_i \in T', \exists \mathcal{M}(a_i) \in Q \quad (5)$$

- 1154 • *Type Preservation:*  $q$  must preserve the ambiguity types of the original action nodes:

$$\tau(\mathcal{M}(a_i)) = \tau(a_i), \quad \forall a_i \in T' \quad (6)$$

1155 where  $\tau(a_i)$  is the ambiguity type of action node  $a_i$ .

- 1156 • *Boundedness:*  $q$  should not introduce any actions outside of  $T'$ :

$$\forall \text{expression } e \in Q, \exists a_i \in T' : e = \mathcal{M}(a_i) \quad (7)$$

1157 **Solution Overview.** We leverage an LLM-based **Text Query Generator** to integrate the ambigu-  
 1158 ties introduced in  $T'$  into a single and coherent query  $q$ , ensuring that the generated query faithfully  
 1159 reflects all the intended ambiguous components. Finally, an **Text Query Verifier** is employed  
 1160 to validate that  $q$  accurately captures the ambiguity without introducing any extraneous semantics.  
 1161 This two-step process—generation followed by verification—ensures that the final query remains  
 1162 consistent with the design decisions encoded in  $T'$  while meeting the criteria of completeness, type  
 1163 preservation, and boundedness.

1164 **Text Query Diversity in Generation.** NLV Corpus [17] defines several distinct categories of natural  
 1165 language utterances—question, command, query, and other. Since “query” somewhat overlaps with  
 1166 other styles, we focus on three main types: question, command, and caption, each representing a  
 1167 distinct style of user input:

- 1168 • *Question:* Typically begins with a question word (e.g., “What”, “How much”, “How many”, etc.).
- 1169 • *Command:* Usually an imperative sentence (e.g., “Show a bar chart of sales by region”).
- 1170 • *Caption:* Includes non-standard phrases, incomplete sentences, or informal text conveying user  
 1171 intent, often brief (e.g., “SUM (Sales) vs Date” or “budget over time”).

1172 To ensure diversity of the generated queries, we provide specific Text Query styles and corresponding  
 1173 example queries as input to the language model. These examples are randomly sampled from a large  
 1174 corpus to ensure variability.

1175 **Text Query Generator.** To systematically align the structured visualization tree with diverse natural  
 1176 language expressions, we define explicit input-output mappings. The input to the LLM (GPT-4o-mini-  
 1177 turbo) delivers essential context, including data schema, sample data, action sequences, and style  
 1178 requirements. This aims to ensure that the output text query: maintains linguistic grounding for all

Table 4: Chart types, visual channels, and analytic tasks with compatible data types:  $C$ =Categorical,  $Q$ =Quantitative,  $T$ =Temporal,  $\emptyset$ =N/A.

Chart Type	Encoding Channel $x y color size theta$	Analytic Task
Bar	$\{C, Q, T\} Q C \emptyset \emptyset$	Trend, Distribution
Line	$\{C, Q, T\} Q C \emptyset \emptyset$	Trend, Distribution
Pie	$\emptyset \emptyset C \emptyset Q$	Distribution
Scatter	$Q Q C Q \emptyset$	Correlation
Heatmap	$\{C, Q, T\} \{C, Q\} Q \emptyset \emptyset$	Correlation
Boxplot	$\{C\} Q C \emptyset \emptyset$	Distribution

actions (5), preserves ambiguity types during translation (6), and avoids introducing any extraneous semantics (7). The complete prompt format is in Table 7.

**Text Query Verifier.** As indicated by recent studies [57, 58], LLMs outputs still require verification, particularly concerning *boundedness* (7). The verification can be performed by LLMs or human evaluators. In our preliminary experiments, we found that LLM-based verification is sufficient to achieve an accuracy of 99%. Thus, we designed the following prompt for LLM verification as shown in Figure 9

If  $L_1$  fully covers all nodes in  $T$  while  $L_2$  remains empty, the  $q$  is considered valid and added to the dataset. Otherwise,  $q$  is classified as invalid, and it would be regenerated by the **Text Query Generator**. This approach checks for *completeness* (5), *type preservation* (6), and *boundedness* (7). If the verification fails, the system can regenerate the query or suggest corrections.

#### B.4 Step 4: Ambiguity-resolved Reasoning Path

Based on the previous discussion, we have reformulated the *Text2VIS* problem from a direct mapping  $q \rightarrow \mathbf{v} = \{v_1, \dots, v_k\}$  to a structured process  $q \rightarrow T' \rightarrow T_{\mathbf{v}}^* \rightarrow \mathbf{v}$ . To mimic human-like reasoning workflow for ambiguity resolution, we propose decomposing the ambiguity-aware visualization generation process into a sequential reasoning path with five distinct steps, as illustrated in Figure 1:

$$q \xrightarrow{\phi_1} S_1 \xrightarrow{\phi_2} S_2 \xrightarrow{\phi_3} S_3 \xrightarrow{\phi_4} S_4 \xrightarrow{\phi_5} \mathbf{v} \quad (8)$$

where each  $\phi_i$  represents a reasoning function and each  $S_i$  represents the intermediate state after applying the corresponding reasoning function.

**Step-①: Data Selection Reasoning.** The first step parses the natural language query  $q$  into data components from the data table:

$$\phi_1(q) \rightarrow S_1 = \{a_1^c, a_2^c, \dots, a_m^c\} \quad (9)$$

where each  $a_i^c$  represents a data component selection action, including column selection, value selection, and filter condition specification. The outcomes of this step correspond to the SELECT and FILTER nodes in the visualization tree (see Figure 6).

**Step-②: Chart Type Reasoning.** The second step determines appropriate visualization mark types based on the analytic task:

$$\phi_2(S_1, q) \rightarrow S_2 = S_1 \cup \{a_1^v, a_2^v, \dots, a_n^v\} \quad (10)$$

where each  $a_i^v$  represents a visualization design action, including analytic task identification and chart type selection. As shown in Table 4, existing visualization design principles [59–61] can establish a mapping relationship between tasks and chart types [59, 60, 62]. When the text query  $q$  does not

1207 explicitly specify a chart type, the identified task can guide inference, though this may introduce  
 1208 ambiguity as multiple chart types may be suitable for a given task. In addition, certain tasks influence  
 1209 encoding channel selection in the next step.

1210 **Step-③: Channel Mapping Reasoning.** The third step establishes the mappings between data  
 1211 components and encoding channels:

$$\phi_3(S_2) \rightarrow S_3 = S_2 \cup \{a_1^m, a_2^m, \dots, a_p^m\} \quad (11)$$

1212 where each  $a_i^m$  represents a channel mapping action, such as assigning data columns to encoding  
 1213 channels like X, Y, color, or size. This step ensures that data columns are mapped appropriately,  
 1214 aligning with visualization design principles, where some mapping relationships are shown in Table 4.

1215 **Step-④: Data Transformation Reasoning.** The fourth step specifies necessary data transformations  
 1216 based on the channel mappings:

$$\phi_4(S_3) \rightarrow S_4 = S_3 \cup \{a_1^t, a_2^t, \dots, a_r^t\} \quad (12)$$

1217 where each  $a_i^t$  represents a data transformation action, including aggregation, binning, sorting, and  
 1218 filtering operations, these transformations prepare the data to be properly visualized according to the  
 1219 selected chart type and channel mappings.

1220 **Step-⑤: Visualization Synthesis Reasoning.** The final step is to integrate all reasoning steps to  
 1221 generate a set of valid visualizations:

$$\phi_5(S_4) \rightarrow \mathbf{v} = \{v_1, v_2, \dots, v_k\} \quad (13)$$

1222 where each  $v_i$  represents a valid visualization specification. This process can produce multiple valid  
 1223 visualizations that address different aspects of the ambiguity in the original query (see Figure 1).

1224 The four reasoning steps (Step-① to Step-④) outlined in the ambiguity-resolved reasoning path are  
 1225 not strictly bound by a fixed sequence and can be executed in any order, provided all steps are  
 1226 completed before the final visualization synthesis (Step-⑤). This flexibility arises because each step  
 1227 addresses a distinct aspect of the visualization process—data selection, chart type reasoning, channel  
 1228 mapping, and data transformation—and their interdependencies are managed through the shared  
 1229 ambiguity-aware visualization tree  $T'$ . The exact order may vary depending on the text query; for  
 1230 example, if the query lacks any clues about the chart type, chart type reasoning may occur last, after  
 1231 all selected data and possible channel mappings have been considered.

1232 This structured reasoning process systematically addresses ambiguity at each step while adhering  
 1233 to visualization design principles. Each step builds upon prior decisions, progressively refining the  
 1234 visualization specifications to account for multiple valid interpretations of the original text query.

1235 Formally, the complete reasoning path can be expressed as the composition of the step-wise reasoning  
 1236 functions:

$$\mathcal{F}(q, D) = (\phi_5 \circ \phi_4 \circ \phi_3 \circ \phi_2 \circ \phi_1)(q, D) \rightarrow \mathbf{v} \quad (14)$$

1237 This decomposition simplifies the ambiguity-aware *Text2VIS* process, breaking down complex  
 1238 reasoning into steps that better align with LLMs' strengths in natural language understanding and  
 1239 generation. Techniques like chain-of-thought prompting or step-wise direct preference optimization  
 1240 (step-DPO) [23, 44] can further improve LLM performance.

1241 Finally, as shown in Figure 2 (d), the LLM-based step-wise reasoning generator takes the text query  
 1242  $q$ , the generated unambiguous visualization  $v$ , and the ambiguous visualization tree  $T'$  as input. It  
 1243 then performs reverse reasoning for each step (14), generating text-based reasoning descriptions.  
 1244 For example, when resolving chart type ambiguity in Figure 1, the LLM reasons, "Since this query  
 1245 requests a trend analysis over time, either bar charts or line charts would be appropriate, as both  
 1246 effectively represent temporal patterns in the data" for Step-②. The complete prompt format is in  
 1247 Table 9.



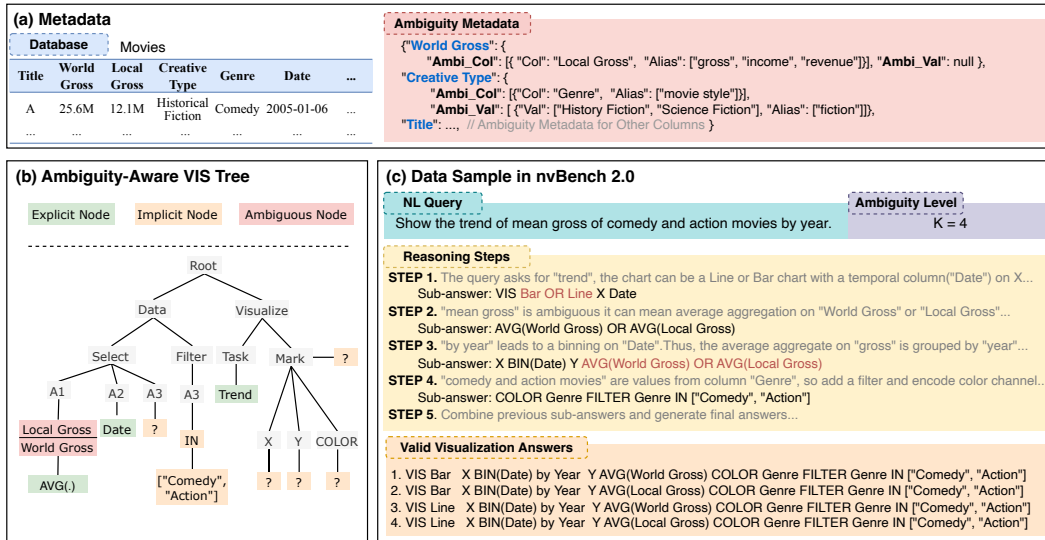


Figure 8: An example in nvBench 2.0.

## C More Details of nvBench 2.0

### C.1 Detailed Example in nvBench 2.0

Figure 8 illustrates an example sample in the nvBench 2.0, showcasing how the data is stored and the information it contains. Figure 8 (a) Presents the data schema for the "Movies" database, incorporating ambiguity metadata that highlights potential ambiguities, such as column aliases (e.g., "World Gross" and "Local Gross" both linked to "gross") and value aliases (e.g., "History Fiction" and "Science Fiction" both linked to "fiction").

Figure 8 (b) Displays the Ambiguity-Aware VIS Tree, depicting the hierarchical structure of ambiguous user intent, with explicit nodes shown in green, implicit nodes in yellow, and ambiguous nodes in red, revealing the underlying ambiguous intents for the data sample in (c).

Figure 8 (c) represents a data sample within nvBench 2.0, beginning with the text query "Show the trend of the mean gross of comedy and action movies by year" and an ambiguity level of  $K = 4$ , demonstrating the number of gold answers available for this query. The sample includes reasoning steps, each with 1-2 sentences of logical reasoning and a sub-answer, culminating in the four gold answers.

### C.2 Detailed Statistics of nvBench 2.0

**Data Tables.** Figure 9 (a.1) shows that most tables in our dataset have 2–5 columns, with fewer than 50 tables having more than 8 columns. As Figure 9 (a.2) illustrates (log scale), row counts range widely, from 10–1000 rows for many tables to outliers exceeding 10,000 rows. This variety ensures that nvBench 2.0 tests system performance across both small and large datasets.

**Ambiguity Types and Levels.** An important contribution of nvBench 2.0 is the systematic introduction of controlled ambiguity levels. Figure 9 (b.1) categorizes ambiguity by type: Data Transformation (DT) ambiguities are most prevalent ( $\sim 3,500$  examples), followed by Channel Mapping (CM) ambiguities ( $\sim 1,500$  examples), with Data Selection (DS) and Chart Type Selection (CT) ambiguities represented by approximately 900 and 400 examples, respectively. As shown in Figure 9 (b.2), the majority of samples (approximately 3,500) have an ambiguity level of 2, indicating that two valid visualizations exist. The dataset also contains a substantial number of samples with ambiguity levels of 3, 4, and 5, enabling a thorough evaluation of systems under increasingly complex ambiguous scenarios. Figure 9 (c.2) and (d.2) further illustrates the relationship between ambiguity levels and two factors: chart types (c.2) and NL styles (d.2), showing comprehensive data coverage.

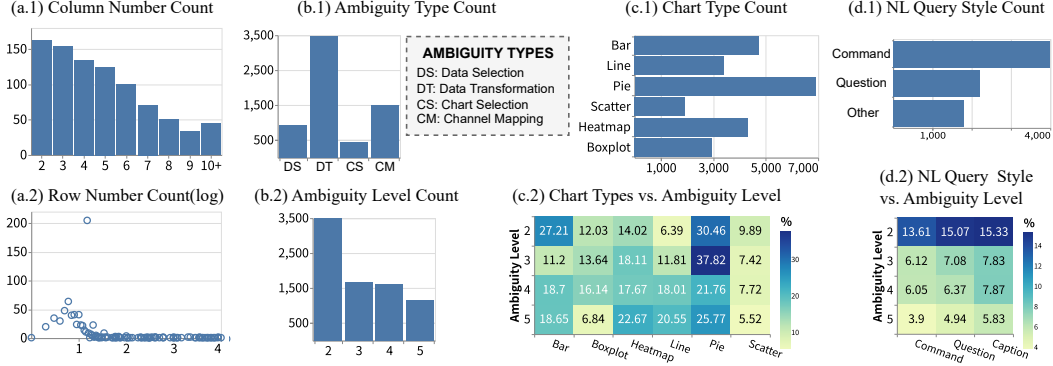


Figure 9: Detailed Statistics of nvBench 2.0.

Table 5: Distribution of natural language styles across chart types and word count statistics

NL Style	Count by Chart Type						Total	Word Count		
	Bar	Line	Pie	Scatter	Boxplot	Heatmap		Avg.	Max	Min
Command	1368	922	1922	608	1319	894	2338	14.20	60	6
Question	1570	1084	2299	679	1403	966	2636	14.04	39	5
Caption	1779	1363	2651	581	1589	1079	2904	14.00	65	5
Total	4717	3369	6872	1868	4311	2939	7878	14.07	65	5

**Visualizations.** Figure 9 (c.1) shows the distribution of chart types in nvBench 2.0. Pie charts are the most common, with around 6,000 examples, followed by bar charts ( $\sim 4,000$ ) and heatmaps ( $\sim 3,500$ ). Additionally, line charts ( $\sim 2,800$ ), boxplots ( $\sim 2,000$ ), and scatter plots ( $\sim 1,500$ ) are also well-represented, ensuring that the benchmark covers all major visualization types. This distribution reflects common visualization practices, where pie and bar charts are widely used for categorical comparisons, while the other types serve specialized analytical needs.

**Text Queries.** Figure 9 (d.1) presents the natural language query distribution. Command-based queries (e.g., “Show me the sales by region”) are most frequent ( $\sim 4,000$ ). Question-based queries (e.g., “What are the sales trends?”) and caption-like statements (e.g., “SUM (Sales) vs Date”) appear in about 2,000 and 1,800 instances, respectively. Table 5 provides a detailed breakdown of NL styles across different chart types, along with word count statistics. Commands, questions, and captions are distributed across various chart types, with pie charts receiving the highest number of queries (6,872). The average word count remains consistent ( $\sim 14$  words), with captions exhibiting the longest maximum length (65 words). This distribution highlights the dataset’s diversity in both linguistic structure and visualization needs, ensuring that nvBench 2.0 can effectively evaluate systems’ capabilities to handle diverse user interactions.

## D More Details of Experimental setups

**Methods.** We evaluate the performance on ambiguous *Text2VIS* tasks using both prompting-based and fine-tuning-based methods with our nvBench 2.0. The primary goal is to assess the model’s ability to generate diverse and semantically accurate visualizations in response to ambiguous TEXT queries.

*Prompting-based Methods.* We evaluate two prompting strategies with GPT-4o-mini, GPT-4o and Qwen2.5-7B-Instruct model:

- **Direct Prompting:** See Table 10 for complete prompt structure. The model receives structured *Data Information* and an *Text Query* as input, generating 1-5 distinct charts to cover possible interpretations of ambiguous queries.

- **Step Prompting:** See Table 11 for complete prompt structure. Models are guided to “think step-by-step”, explicitly articulating their reasoning process before generating visualizations. Models using this approach are denoted with a “-Step” suffix.

#### 1307 Supervised Fine-tuning Method.

- **Qwen2.5-7B-SFT:** We performed supervised fine-tuning on the Qwen2.5-7B-Instruct model using the training set, enabling direct generation of multiple Vega-Lite definitions without step-wise reasoning. Training involved three epochs with a global batch size of 16, a learning rate of  $2e-5$ , the AdamW optimizer, and a cosine learning rate scheduler with a 0.1 warmup ratio.

#### 1312 Preference Learning Method.

- **Step-Text2Vis:** We designed Step-Text2Vis to handle the ambiguity in *Text2VIS* through step-wise reasoning as detailed in Section 3. After the initial supervised fine-tuning of Qwen-2.5-7B-Instruct, we constructed a preference dataset from the nvBench 2.0 development set for Step-DPO training. This process used one epoch with a global batch size of 4, a linearly decaying learning rate from  $2e-6$ , and the AdamW optimizer.

1318 **Evaluation Metrics.** Detailed explanation for evaluation metrics:

- **Precision@K (P@K):** Assesses recommendation accuracy by calculating the proportion of valid visualizations among the top-K outputs. Higher P@K indicates more trustworthy recommendations, with fewer incorrect visualizations shown to users.
- **Recall@K (R@K):** Quantifies how completely the model covers the golden visualization space by measuring the proportion of valid visualizations successfully identified. This captures the model’s ability to represent multiple valid interpretations for ambiguous queries.
- **F1@K:** Provides a balanced measure that combines precision and recall through their harmonic mean. This comprehensive metric rewards systems that achieve both high coverage of the golden answer space and high accuracy in their recommendations.

1328 For all experiments, we report these metrics at  $K \in \{1, 3, 5\}$  to evaluate performance across different  
1329 recommendation set sizes.

## 1330 **E More Details of Error Analysis**

### 1331 **E.1 Qualitative Error Analysis in Reasoning Paths.**

1332 Figure 10 outlines frequent errors in the stepwise reasoning process for ambiguous Text2VIS tasks  
1333 across four key steps:

- At the Data Selection step, errors frequently arise when models overgeneralize column references, selecting irrelevant or overly broad sets of data that do not align with the query’s intent.
- During the Data Transformation step, mistakes often occur due to misinterpretation of aggregation requirements, such as applying incorrect operations like counting rows instead of summing specific values.
- At the Chart Selection step, errors commonly stem from choosing inappropriate chart types that fail to match the analytical task or data characteristics, leading to ineffective visualizations.
- In the Channel Mapping step, issues frequently emerge from incorrect assignments of data to visualization channels, such as mapping a temporal field to a color channel instead of an axis, resulting in misleading representations.

1344 The stepwise reasoning step information provided in these examples is valuable for further ambiguous  
1345 *Text2VIS* system development, as it offers insights into common reasoning pitfalls, enabling the  
1346 design of more robust models that can learn from and correct these errors to improve accuracy and  
1347 interpretability in handling ambiguous queries.

Example 1: Error at Data Selection Step																			
Text Query	<div>Database: Stores</div> <table><tr><th>workshop_id</th><th>store_id</th><th>store_name</th><th>email_address</th><th>...</th></tr><tr><td>1</td><td>A01</td><td>Store A</td><td>a@xx.com</td><td>...</td></tr><tr><td>2</td><td>B01</td><td>Store B</td><td>b@xx.com</td><td>...</td></tr></table>				workshop_id	store_id	store_name	email_address	...	1	A01	Store A	a@xx.com	...	2	B01	Store B	b@xx.com	...
workshop_id	store_id	store_name	email_address	...															
1	A01	Store A	a@xx.com	...															
2	B01	Store B	b@xx.com	...															
"The distribution of ids shown in a pie chart."																			
Bad Reasoning Step	<div>Reasoning:</div> The query mentions "distribution of ids" and a "pie chart." "Ids" likely refers to any ID-like column, so I'll select workshop_group_id, address_id, store_name, and store_email_address. <div>Sub-Answer:</div> SELECT [workshop_group_id, address_id, store_name, store_email_address]. <div>Problem:</div> Overgeneralizes "ids" to include non-ID columns, leading to incorrect answers.																		
Good Reasoning Step	<div>Reasoning:</div> The query mentions "distribution of ids" and a "pie chart." "Ids" could refer to workshop_group_id or address_id, not store_name or store_email_address. <div>Sub-Answer:</div> SELECT [workshop_group_id, address_id].																		

Example 2: Error at Data Transformation Step																			
Text Query	<div>Database: Player</div> <table><tr><th>player</th><th>team</th><th>game_played</th><th>season</th><th>...</th></tr><tr><td>Alice</td><td>Team A</td><td>25</td><td>2024</td><td>...</td></tr><tr><td>Bob</td><td>Team B</td><td>30</td><td>2024</td><td>...</td></tr></table>				player	team	game_played	season	...	Alice	Team A	25	2024	...	Bob	Team B	30	2024	...
player	team	game_played	season	...															
Alice	Team A	25	2024	...															
Bob	Team B	30	2024	...															
"Show the distribution of games played by team in a bar chart."																			
Bad Reasoning Step	<div>Reasoning:</div> The query asks for "distribution of games played by team." I'll aggregate by counting rows for each Team to show distribution. <div>Sub-Answer:</div> SELECT COUNT(*) GROUP BY team <div>Problem:</div> Incorrectly applies a count of rows, ignoring that game_played already represents game counts, leading to player counts per team.																		
Good Reasoning Step	<div>Reasoning:</div> The query requests "distribution of games played by team." Game_Played is already a count, so I'll aggregate it (sum or average) for each Team. <div>Sub-Answer:</div> SELECT SUM(game_played) GROUP BY team																		

Example 3: Error at Chart Selection Step																			
Text Query	<div>Database: Product</div> <table><tr><th>category</th><th>sales</th><th>region</th><th>date</th><th>...</th></tr><tr><td>X</td><td>5M</td><td>North</td><td>2024-01-15</td><td>...</td></tr><tr><td>Y</td><td>7.5M</td><td>South</td><td>2024-02-20</td><td>...</td></tr></table>				category	sales	region	date	...	X	5M	North	2024-01-15	...	Y	7.5M	South	2024-02-20	...
category	sales	region	date	...															
X	5M	North	2024-01-15	...															
Y	7.5M	South	2024-02-20	...															
"Distribution of total sales in 2024 by product category."																			
Bad Reasoning Step	<div>Reasoning:</div> The query mentions "distribution" of total sales by product category . Pie and bar charts work, and a boxplot might show distribution. <div>Sub-Answer:</div> VIS [Bar, Pie, Boxplot]. <div>Problem:</div> Although boxplot is good for distribution of multiple values within categories, it is unsuitable for single values per department,																		
Good Reasoning Step	<div>Reasoning:</div> The query indicates "distribution" of total sales by product category with one value per category. Pie and bar charts are valid, but not boxplot. <div>Sub-Answer:</div> VIS [Bar, Pie].																		

Example 4: Error at Channel Mapping Step																			
Text Query	<div>Database: Product</div> <table><tr><th>category</th><th>sales</th><th>region</th><th>date</th><th>...</th></tr><tr><td>X</td><td>5M</td><td>North</td><td>2024-01-15</td><td>...</td></tr><tr><td>Y</td><td>7.5M</td><td>South</td><td>2024-02-20</td><td>...</td></tr></table>				category	sales	region	date	...	X	5M	North	2024-01-15	...	Y	7.5M	South	2024-02-20	...
category	sales	region	date	...															
X	5M	North	2024-01-15	...															
Y	7.5M	South	2024-02-20	...															
"Visualize sales and region data over time in a bar chart."																			
Bad Reasoning Step	<div>Reasoning:</div> The query mentions "sales" and "region" over time. I'll map Sales to Y, Region to X, and Date to color. <div>Sub-Answer:</div> X region, Y sales, COLOR Date <div>Problem:</div> Although temporal column "Date" can be mapped to COLOR in a scatterplot, it should be mapped to X in a bar chart.																		
Good Reasoning Step	<div>Reasoning:</div> The query specifies "sales" and "region" over time. Sales to Y, Date to X, and Region to color. <div>Sub-Answer:</div> X date, Y sales, COLOR region																		

Figure 10: Examples of Stepwise Reasoning Errors in the nvBench 2.0 dataset, highlighting common pitfalls in the Text-to-Visualization process.

Table 6: Licenses List for Assets Used

Asset	Usage	License
GPT-4o [63]	Baselines and query verification	Custom License
GPT-4o-mini	Baselines, metadata generation, query synthesis	Custom License
Qwen2.5-7B [64]	Baselines and model fine-tuning	Apache-2.0
nvBench Dataset [16]	Source for data tables and seed visualizations	MIT
BIRD Dataset [21]	Source for additional data tables	CC BY-SA 4.0
ConceptNet [53]	Semantic alias discovery in ambiguity metadata	CC BY-SA 4.0
ASP Solver (Clingo) [22]	Visualization query resolution	MIT

## F Limitations

While nvBench 2.0 introduces significant advancements in ambiguity-aware Text2VIS benchmarking, several limitations remain that present opportunities for future research:

**Limited Coverage of Visualization-Adjacent Tasks:** Although our benchmark focuses on Text2VIS ambiguity resolution, it does not extend to related domains such as Text2SQL. The step-wise reasoning approach could potentially be adapted to handle SQL generation with ambiguous queries, particularly since visualization and database queries share similar data operations. Future work could explore the integration of both Text2VIS and Text2SQL ambiguity resolution within a unified framework.

**Restricted Chart Types and Components:** Though nvBench 2.0 includes six chart types (bar, line, pie, scatter, heatmap, and boxplot), it does not cover the full spectrum of visualization techniques. Advanced chart types like treemaps, network diagrams, geographic maps, and multi-view coordinated visualizations are not included. Additionally, the benchmark lacks support for more sophisticated chart components such as error bars, trend lines, annotations, interactive elements, and customizable legends that are often crucial for comprehensive data storytelling.

**Limited Integration with Statistical Analysis:** The current benchmark treats visualization as the primary goal rather than integrating it with deeper statistical analysis intents. Users often request visualizations to support specific analytical objectives (hypothesis testing, correlation analysis, outlier detection, clustering) that require a tighter coupling between visualization and statistical computation. Future work could expand the benchmark to include cases where visualization serves as a component within broader analytical workflows.

**Absence of Conversational Context:** nvBench 2.0 evaluates standalone queries without considering the conversational context in which they might appear. In real-world scenarios, visualization requests often occur within multi-turn dialogues where context from previous exchanges influences interpretation. The benchmark does not account for these contextual dependencies, limiting its ability to evaluate systems in realistic interactive settings where ambiguity resolution might span multiple conversational turns.

## G Ethic Statement

This paper introduces nvBench 2.0, a novel benchmark for ambiguous Text-to-Visualization tasks, and evaluates the capabilities of LLMs in resolving visualization ambiguity. Our work is not intended to provoke anxiety, but rather to gain a better understanding of how LLMs can be leveraged to interpret ambiguous natural language queries for data visualization. This study aims to foster discussions on how visualization systems can better accommodate the inherent ambiguity in human requests, and how humans can more effectively utilize LLM-powered visualization tools.

In developing nvBench 2.0, we have ensured that the dataset does not contain sensitive or personally identifiable information. The data tables used in our benchmark are derived from public datasets (nvBench [20] and BIRD [21]) and have been carefully reviewed to exclude potentially harmful or private content.

In our experimental evaluations involving human reviewers for query verification, participants received appropriate compensation and ensured adequate rest periods between review sessions. We rigorously

Table 7: Prompt Structure for Ambiguous Text Query Synthesis

Prompt Structure for Ambiguous Text Query Synthesis	
<b>### Task Description:</b>	
You are an intelligent assistant. You will create three text queries (command, question, and caption) based on a given data schema and action list. Each query must incorporate all information from the action list without introducing extra elements.	
<b>### Process (4 steps):</b>	
# Step 1: Interpret Visualization Type. <i>e.g.</i> , ...	
# Step 2: Rephrase Data Columns. <i>e.g.</i> , ...	
# Step 3: Rephrase Data Transformations. <i>e.g.</i> , ...	
# Step 4: Rephrase Filter Conditions. <i>e.g.</i> , ...	
<b>### Final Answer Construction:</b>	
Combine the rephrased elements from steps 1-4 to create three text queries:	
1. Command-style: Direct instruction ( <i>e.g.</i> , "Plot the total sales by month for products priced over \$50")	
2. Question-style: Inquiry format ( <i>e.g.</i> , "What was the average rating for movies released during 2020?")	
3. Caption-style: Declarative format ( <i>e.g.</i> , "Distribution of customer count by region in a pie chart.")	
<b>### Input</b>	
Database: {basename}	
Data Columns: {data_schema}	
Data Value Examples: {data_value_example}	
Ambiguous Column Pairs: {ambiguous_pairs}	
Action List: {action_list}	

1387 protect participants' personal information, ensuring that their information remains confidential and is  
1388 not disclosed in this document or the GitHub repository.

1389 Our source code and data are under GPL-3 license, and we follow the licenses of assets used in this  
1390 paper, as listed in Table 6.

Table 8: Prompt Structure for Text Query Verification

Prompt Structure for Text Query Verification
<p><b>### Task Description:</b></p> <p>You are a smart assistant. Your job is to check if a text query for a Text-to-Visualization (Text2VIS) task is valid. The query must match all parts of the provided visualization tree (T) and follow three rules:</p> <ol style="list-style-type: none"> <li>1. <b>Completeness:</b> The query must include all actions and elements from the visualization tree, such as data selections, transformations, chart types, and channel mappings.</li> <li>2. <b>Type Preservation:</b> The query must keep the same ambiguity types (e.g., Data Transformation, Channel Mapping, Data Selection, Chart Type Selection) as defined in the visualization tree.</li> <li>3. <b>Boundedness:</b> The query must not add extra information or elements not in the visualization tree.</li> </ol> <p>You will get the data schema, action list, visualization tree (T), and the text query (q). Verify if the query meets all three rules. If the query is invalid, explain what is missing or extra to help fix it.</p>
<p><b>### Input:</b></p> <p># <b>Database:</b> [baseline]</p> <p># <b>Data Schema:</b> {data_schema}</p> <p># <b>Data Value Examples:</b> {data_value_example}</p> <p># <b>Visualization Tree (T):</b> {vis_tree}</p> <p># <b>Synthesized Text Query (q):</b> [text_query]</p>
<p><b>### Output:</b></p> <p># <b>Validity:</b> [Valid/Invalid]</p> <p># <b>Completeness:</b> [Met/Not Met] - Confirm if the query includes all actions and elements from the visualization tree (T).</p> <p># <b>Type Preservation:</b> [Met/Not Met] - Confirm if the query preserves the ambiguity types as defined in the visualization tree.</p> <p># <b>Boundedness:</b> [Met/Not Met] - Confirm if the query avoids adding extra information not in the visualization tree.</p> <p># <b>Feedback (if Invalid):</b> Explain what makes the query invalid, including missing elements or extra information, to guide fixing or regenerating the query.</p>

Table 9: Prompt Structure for Step-wise Reasoning Synthesis

Prompt Structure for Step-wise Reasoning Synthesis
<b>### Task Description:</b> You are a good data visualization expert. Given an ambiguous/incomplete Text Query with Data Schema, and the step-by-step answer recommending visualization charts corresponding to the ambiguous/incomplete Text Query. Then, you need to fill in the reasoning process for each step.
<b>### Instructions:</b> <b># Step 1: Data Selection.</b> Select data columns and data filters mentioned in the Text Query. If the Text Query is ambiguous and can be mapped to multiple columns, use a list to indicate all columns. <b># Step 2: Data Transformation.</b> Select data transformation (operation : parameter) = (aggregate : [sum, mean, count]; bin : base; sort:[ascending, descending]) mentioned in the Text Query. <b># Step 3: Chart type Selection.</b> Select all valid chart types for visualization based on Text Query and data selected. If chart type indicated in the Text Query, select from chart mark=(bar, line, arc, point, rect, boxplot). Else if no chart type mentioned, but specific analysis task mentioned in the Text Query, inference chart type by (task:chart)=(trend:[bar,line]; distribution:[bar,arc,line,boxplot]... Also consider if the chart type can visualize selected data. <b># Step 4: Selected Column-Channel Mapping.</b> Map selected data columns to encoding channels = (x, y, color, size). You should consider basic channel mapping feasibility. Answer with all valid chart-channel-column mapping solutions. <b># Step 5: Visualization Synthesis.</b> Based on previous steps, synthesis the final visualizations.
<b>### Input:</b> Database: {basename} Data Columns: {data_schema} Data Value Examples:{data_value_example} Text Query: {text_query}
<b>### Output</b> # Step 1: <reasoning>...</reasoning> <answer> ... <answer> # (Step 2-4) # Step 5: <answer> [ VIS 1, VIS 2, ..., VIS k ] <answer>

Table 10: Prompt Structure for Basic Experiments

Prompt Structure for Basic Experiments
<b>### Task Description:</b> You are a good data visualization expert. Given an ambiguous/incomplete Natural Language Query and a Data Table, please recommend 1 to 5 different charts corresponding for the ambiguous/incomplete NL Query. Please strictly follow the output format.
<b>### Example:</b> <b># Input:</b> Database: {basename} Data Columns: {data_schema} Data Value Examples: {data_value_example} Query: {text_query} <b># Output:</b> <answer> [ VIS 1, VIS 2, ..., VIS k ] </answer>



Table 11: Prompt Structure for Stepwise Reasoning Experiments

Prompt Structure for Stepwise Reasoning Experiments
<p><b>### Task Description:</b></p> <p>You are a good data visualization expert. Given an ambiguous/incomplete Natural Language Query and a Data Table, please recommend 1 to 5 different charts corresponding for the ambiguous/incomplete NL Query.</p> <p>Please think step by step and strictly follow the output format.</p>
<p><b>### Example:</b></p> <p><b># Input:</b></p> <p>Database: {basename}</p> <p>Data Columns: {data_schema}</p> <p>Data Value Examples: {data_value_example}</p> <p>Query: {text_query}</p> <p><b>### Output:</b></p> <p># Step 1: &lt;reasoning&gt;...&lt;/reasoning&gt; &lt;answer&gt; ... &lt;/answer&gt;</p> <p># (Step 2-4)</p> <p># Step 5: &lt;answer&gt; [ VIS 1, VIS 2, ..., VIS k ] &lt;/answer&gt;</p>