Interactive Text-to-Visualization: Refining Visualization Outputs Through Natural Language User Feedback

Xubang Xiong
The Hong Kong University of Science and Technology
Hong Kong, China
xxiongag@connect.ust.hk

Raymond Chi-Wing Wong*
The Hong Kong University of Science and Technology
Hong Kong, China
raywong@cse.ust.hk

Yuanfeng Song*
AI Group, WeBank Co., Ltd
Shenzhen, China
songyf@connect.ust.hk

Abstract

Data visualization (DV) is of significance to the data analysis applications, exploring the hidden patterns and showing the insightful data. The task of *Text-to-Vis*, which takes text as input and generates data visualizations, was proposed to lower the threshold for generating DVs. However, the existing methods only view this problem as a one-shot mapping problem, directly outputting the final DVs without considering any user feedback for refining the generated DVs. Motivated by this, a more interactive scenario is investigated, where users could provide natural language feedback to refine the generated DVs. The scenario is formulated as the *Text-to-Vis with Feedback* problem. A new dataset is also created to further study the problem, which contains the user utterance, database schema, generated DVs, the natural language feedback and the refined DVs. A large language model (LLM) based framework named *Vis-Edit* is designed for handling this task, including schema linking, clause location, clause generation, merger and self-consistency. Eventually, extensive experiments reveal the effectiveness of *Vis-Edit*.

CCS Concepts

• **Human-centered computing → Visualization systems and tools**.

Keywords

data visualization, natural language interface, large language model

ACM Reference Format:
Xubang Xiong, Raymond Chi-Wing Wong, and Yuanfeng Song. 2025. 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 (CIKM '25), November 10–14, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

1 Introduction

Data visualization (DV) is important to the database [19, 23] and the data mining community [20, 30, 36, 37], since it is beneficial to reveal hidden patterns. For example, a study presented at KDD'21 [30] investigated an end-to-end DV recommendation system.

Conducting DVs from scratch is difficult for the users who lack the background knowledge of visualization tools or programming skills. To construct the DVs, users should specify their DV requirements (e.g., chart type) to compose the *visualization specifications*, following the high-level grammar of *declarative visualization languages (DVLs)*, a simple language proposed to represent the complex combinations of visualization specifications. There are a great number of DVLs in the community, such as Vega-Lite [32], ggplot2 [40], ZQL [35], ECharts [15], and VizQL [8]. It is hard for the users to master all the DVLs, due to the diversity in grammars and syntaxes.

The task of *Text-to-Vis* [23] is proposed to lower the barrier to conduct DVs. Inspired by the success of natural language interfaces (NLIs), the task of *Text-to-Vis* is to generate DVs given the natural language utterances and the corresponding database schema. Specifically, the given natural language utterance and the data schema are transformed into a data visualization query (DVQ), e.g., Vega-Lite. The DVQ is then converted into a visualization specification that is executable to render a bar chart-like DV. However, the existing models (e.g., *Seq2Vis* [23], *ncNet* [24], *DataVisT5* [41] and *prompt4vis* [17]) model the process as a *one-shot mapping* problem. These models only receive user utterances and schema to output the DVQs directly. The DVQs generated by these text-to-vis models may not reflect users' true demand, since the users could not realize their true demand well or could input the erroneous natural language utterance to the model. Besides, it is possible that the model generates incorrect DVs though the input is correctly specified. For example, shown in Figure 1, the database contains two tables named 'allergy_type' and 'has_allergy', respectively. The 'allergy_type' table has two columns, Allergy and AllergyType. The 'has_allergy' table also has two columns, StuID and Allergy. Given these two tables, it is easy for the model to make confusion, since their table names and the corresponding column names are similar.

Motivated by these, the task of *Text-to-Vis with Feedback* is proposed. Given a natural language utterance of user, the whole database, a generated DVQ based on user and a natural language feedback on the generated DVQ, the task is to refine the generated DVQ to meet the requirement of the user. Figure 1 shows the interaction between the user and the proposed system. One user provides the natural language utterance, *"Show all allergies with number of students affected with a bar chart"*, and the whole database to the system. The system generates a visualization form of the DVQ generated by the system with the corresponding explanation, which illustrates how to construct the chart in detail. The explanation of DVQ can be synthesized by rules or powerful neural models,

*Raymond Chi-Wing Wong and Yuanfeng Song are the corresponding authors.

CIKM '25, Seoul, Republic of Korea.

ACM ISBN 979-8-4007-2040-6/2025/11
https://doi.org/10.1145/XXXXXX.XXXXXX

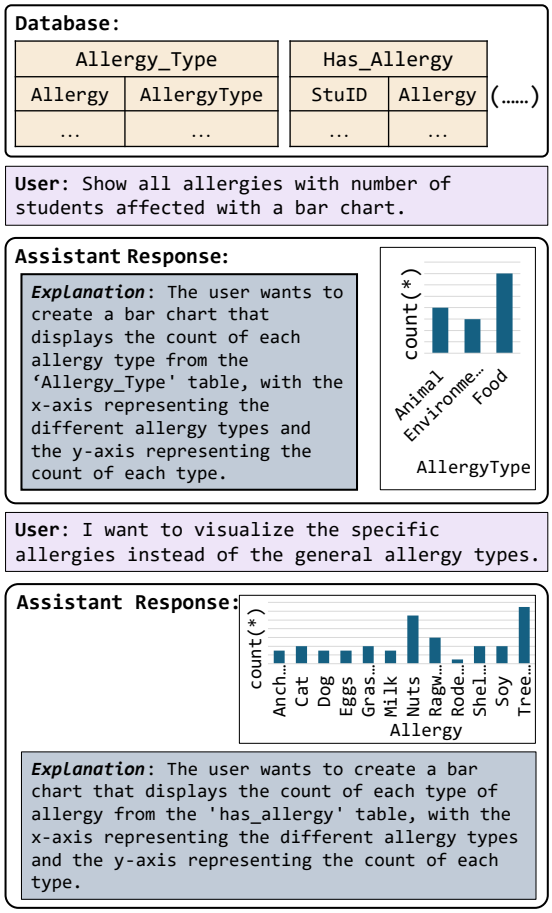

Figure 1: An example of human interaction with a Text-to-Vis system to correct the query of an input utterance.

beneficial to user to further determine whether the DV is correct. In this scenario, the system generates a visualization chart with the wrong schema linking. The system visualizes the data of the AllergyType column from the 'allergy_type' table instead of that of the Allergy column from the 'has_allergy' table. This schema is confusing due to the similar semantic information. To distinguish these tables and columns, human feedback is required to explain the demand of the user. Although the user has no background knowledge of DVQ, it can give natural language feedback to instruct the system to refine the incorrect chart, with the help of the available explanation. In this paper, the task of *Text-to-Vis with Feedback* is modeled as single-round conversation problem, which can be easily expanded to multi-round conversation problem in future work.

A simple solution to utilize the existing large language model (LLM) prompt strategies like Chain-of-Thought (CoT) [43] seems to be effective. However, the simple CoT framework could not make full use of the natural language feedback, even under the few-shot setting, shown in the our experimental results. In this study, a new LLM prompt-based framework named *Vis-Edit* is proposed to enhance the interactive capabilities of general LLMs on *Text-to-Vis with Feedback* task. Specifically, the main idea of the proposed framework is to re-use the generated DVQ with refinement. An inappropriate visualization contains the inappropriate source of data, the inappropriate sorting order, and the inappropriate transformation method. All of them are associated with the clauses of DVQs, since each clause corresponds to one or more visual elements of a visualization chart, discussed later. Hence, the inappropriate visualization could be refined by editing the corresponding clauses, implemented by the *Local Modification System (LMS)* in the proposed framework. The LMS contains three parts, namely the *clause location module*, the *clause generation module* and the *merger*. Given the input text of user, the whole database and the generated DVQ of the inappropriate visualization, the clause location module outputs the inappropriate clauses of the generated DVQ. The clause generation module refines the inappropriate clauses, based on the input of user and the inappropriate clauses detected by the clause location module. The merger is designed to replace the inappropriate clauses in the DVQ with the refined ones. However, there are still two challenges in the framework. One of the challenges is the *schema linking*. For example, the user gives feedback that *"to correct the query, make sure to group the results by the manufacturers name of products"*. The LMS may mistakenly select *Products.Manufacturer* instead of *Manufacturers.Name*, where *Products.Manufacturer* is a set of manufacturers IDs and does not refer to the manufacturers name. These inconsistency between the natural language feedback and database schema make LMS difficult to correct data-related errors. Schema linking module is designed to alleviate this issue. Compared with the schema linking modules of the existing research [6, 28, 31], the designed module not only considers the user utterance, but also leverages user feedback and DVQ to infer the semantic information and predict the potential database schema. Besides, it is a prompt-based method, which also utilizes the implicit knowledge of the LLMs. The other challenge is the problem of *error propagation*. The result of the downstream modules, like the clause generation module, is affected by the upstream modules, like the schema linking module. Wrongly predicted schema is likely to result in mistakenly generated clauses. To improve the robustness of the framework, the self-consistency module is employed, based on the major idea that varied perspectives converging on the same conclusion enhance confidence in its accuracy. This module is widely integrated into many LLM-based frameworks [6, 14], though it may introduce some additional time overhead. Specifically, multiple DVQ candidates generated by the LMS will be used to conduct a majority voting. The DVQ with the highest number of occurrences will be selected as the final answer.

The key contributions of this work are summarized as follows.

- To the best of our knowledge, this is the first work that proposes the *Text-to-Vis with Feedback* problem, applying the natural language feedback to the field of data visualization with the natural language interface.
- A novel dataset construction pipeline was proposed to take advantage of the cooperation between LLM and human evaluation. It not only improves efficiency but also produces high-quality data.
- The proposed LLM-based framework, *Vis-Edit*, is shown to solve the *Text-to-Vis with Feedback* problem effectively.
- Extensive simulations validated the viability of *Vis-Edit* and the analysis of performance and ablation study give guidance to use the framework. In all cases, the overall accuracy of

Vis-Edit surpasses those of all the baselines by at least 7.18% and up to 19.46%.

The rest of the paper is organized as follows. The preliminaries and the new task are introduced in Section 2. The details of dataset construction are explained in Section 3. Section 4 describes the design and analysis of the proposed framework. Section 5 shows the performance of the proposed framework. Finally, Section 6 reviews related work and Section 7 concludes this paper.

2 Preliminaries

In this section, the preliminaries will be introduced to help readers have a better understanding of the following work. The basic concepts (Section 2.1) and the in-context learning (Section 2.2), which will be used in the proposed framework, are formulated. In addition, the formal definition of the new task, *Text-to-Vis with Feedback*, will be given in Section 2.3. The main notations are listed in Table 1.

Table 1: Notations

Notation	Description
q_i	The natural language utterance asked by user i.
$\mathcal{D}$	The whole database schema provided by user i.
p_i'	The inappropriate DVQ generated by the *Text-to-Vis* model based on q_i and $\mathcal{D}$.
p_i	The ground-truth DVQ that meets the requirement of user i.
f_i	The natural language feedback given by user i.

2.1 Basic Concepts

Database Schema. A database schema is the structure that defines how data is organized and stored in a database, including tables, columns, data types, and relationships between tables. In this paper, it is composed of tables $\boldsymbol{T} = \{t_1, \ldots, t_{|T|}\}$. Each table t_i contains several columns $\boldsymbol{C_i} = \{c_1, \ldots, c_{|C_i|}\}$, primary keys $\boldsymbol{M_i} = \{m_1, \ldots, m_{|M_i|}\}$ and foreign keys $\boldsymbol{G_i} = \{g_1, \ldots, g_{|G_i|}\}$. The whole database schema is denoted by $\mathcal{D} = (\boldsymbol{T}, \mathcal{C}, \mathcal{M}, \mathcal{G})$, where $\mathcal{C} = \{\boldsymbol{C_1}, \ldots, \boldsymbol{C_{|T|}}\}$, $\mathcal{M} = \{\boldsymbol{M_1}, \ldots, \boldsymbol{M_{|T|}}\}$ and $\mathcal{G} = \{\boldsymbol{G_1}, \ldots, \boldsymbol{G_{|T|}}\}$.

Schema Link. It refers to the process of mapping parts of a natural language utterance to the corresponding columns and tables in a database schema. The output ($\mathcal{D}'$) of the process is a subset of the given database schema, represented by $\mathcal{D}' \subseteq \mathcal{D}$.

Visualization Specification. The visualization specification specifies what and how the users want to visualize, like the chart type, color, size, legend and so on. The visualization specification often follows the high-level grammar, *declarative visualization language*, including the popular tool Vega-Lite [32].

Data Visualization Query. The definition of data visualization query (DVQ) was proposed by [21, 22], which is a simple language designed for capturing all possible visualization specifications. It can be trivially transformed into any visualization specification. Furthermore, the 'VISUALIZE' clause specifies the visualization type, the 'SELECT' clause extracts the data to be visualized, and the 'FROM' clause determines the data source. The 'TRANSFORM' clause is responsible to transform the selected data, including the 'BIN' clause and the 'GROUP BY' clause. Specifically, the 'BIN' clause is used to group continuous data into discrete bins or categories, beneficial to easily create histograms. The 'ORDER BY' clause decides the order of the data to be visualized.

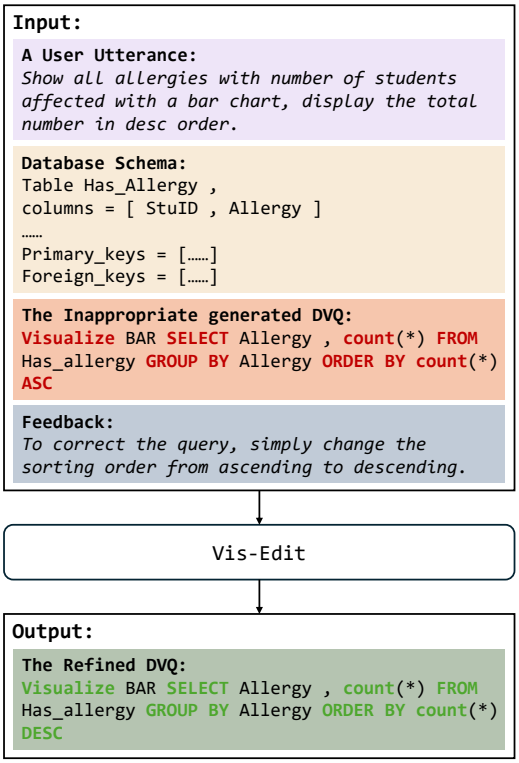

Figure 2: An example of the task of *Text-to-Vis with Feedback*.

2.2 In-Context Learning

Large Language Models (LLMs) [1, 5, 10] have been demonstrated strong performance of downstream tasks, conditioned on an input prompt with a few demonstrations that are used to describe these downstream tasks. It also refers to *In-Context Learning (ICL)*, which aligns the models with the intents of the users, without modifying the weights of the models. The generation process can be formulated as follows.

$$\hat{Y} = LLM(\boldsymbol{P}(X, I, \boldsymbol{E})), \tag{1}$$

where $\hat{Y}$ is the output for a new input X. $\boldsymbol{P}$ is a prompt template, I is the description of a downstream task and $\boldsymbol{E} = \{(x_1, y_1), \ldots, (x_{|E|}, y_{|E|})\}$, where x_i is a sample input and y_i is the corresponding sample output. Moreover, in prompt engineering, *n-shot* refers to the setting that there are n demonstrations in the prompt (i.e., $|\boldsymbol{E}| = n$).

2.3 Task

The task of text-to-visualization with natural language feedback (*Text-to-Vis with Feedback*) is formally defined as follows. Given a natural language utterance q_i of user i, the whole database schema $\mathcal{D} = (\boldsymbol{T}, \mathcal{C}, \mathcal{M}, \mathcal{G})$, an inappropriate DVQ p_i' and a natural language feedback f_i on p_i', the task is to refine the inappropriate DVQ p_i' to an appropriate DVQ p_i. The inappropriate DVQ refers to the DVQ that could not meet the requirement of user. The natural language feedback is a text given by the user to instruct the system to edit the DVQ. The block diagram is shown in Figure 2.

3 Dataset Construction

In this section, the approach for constructing the dataset for our proposed task, *Text-to-Vis with Feedback*, will be described. The inappropriate data visualization queries are generated (Section 3.1), employing the existing *Text-to-Vis* model. These inappropriate queries will be used to synthesize the associated natural language feedback, with the help of the powerful LLMs (Section 3.2). These feedback will be further analyzed in Section 3.3.

3.1 Generating the Inappropriate Data Visualization Query

The public *NVBench* [23] dataset, which is modified based on *Spider* [47], is used as our source of utterances. The incorrect predictions from a text-to-vis model, e.g., Seq2Vis [23], were selected to ensure that our dataset reflects the distribution of errors made by the existing model. The task of generating the inappropriate DVQ is defined as follows. Given a utterance q and the associated database schema $\boldsymbol{D}$, the text-to-vis model generates the DVQs. The DVQs that are inconsistent with the ground-truth are selected as the inappropriate DVQ set.

To expand the inappropriate DVQ set, the *NVBench* dataset was divided into five parts for fivefold cross-validation, with beam search (of size 3) generating multiple predictions to identify model errors. Edit distance via *Levenshtein distance algorithm* [12] was used to analyze discrepancies between inappropriate and refined DVQs, revealing 80% of errors had an edit distance within 4, following the *long-tail* distribution. With *Pareto Analysis* [29], DVQs with edit distance larger than 5 were removed from the NVBench-Feedback dataset, focusing on the critical 80% of errors for model refinement, aligning with prior work [9]. Details of generating the inappropriate DVQ set could be found in Section A.1 of Appendix.

3.2 Generating the Associated Feedback

Large language model (LLM), like Llama3-70B[5], was used to synthesize the natural language feedback, since it has a lot of advantages. For example, a large number of feedback data is generated at low cost with the help of the powerful LLMs. Compared with utilizing the rule-based methods, the synthetic data of LLMs is rich in diversity. The framework contains two parts, explanation generator and feedback generator. The explanation generator explains the inappropriate DVQ and the refined one in a human-understandable way. The feedback generator provides feedback to modify the inappropriate DVQ, given the user utterance, both queries and the corresponding explanations. In addition, some constrains of the output format (e.g., JSON format) were added into the prompt to guarantee the quality of feedback. Details of generating the associated feedback could be found in Section A.2 of Appendix.

The size of *NVBench-Feedback* is 48,482. It is split under the cross-domain setting. That is, the specific domains (e.g., restaurants and flights) of each split are different. In addition, human evaluation was conducted to ensure the quality of the synthesized feedback. Specifically, the authors with expertise in databases assessed whether the feedback could be effectively used to refine the corresponding flawed visualizations. The accuracy of the feedback on the test set was approximately 92.53%.

3.3 Dataset Analysis

Table 2: Dataset Summary

Number of	Train	Validate	Test
Examples	38,393	4,895	5,194
Databases	112	17	22
Uni. Utterance	18,209	2,436	2,591
Uni. Wrong DVQ	14,514	2,634	2,828
Uni. Correct DVQ	4,653	768	863
Uni. Feedback	35,079	4,695	4,969
Avg. Feedback tokens	41.65	41.56	41.88

In this section, the summary of the dataset will be shown. The pattern of the data will be discussed. Table 2 provides a summary of *NVBench-Feedback*. *NVBench-Feedback* is split under the cross-domain setting, according to the ratio of 80/10/10. As a result, the size of training set is 38,393, that of validate set is 4,895 and that of test set is 5,194. The average length of the feedback tokens of all partitions exceeds 40, with tokenization done at the word level.

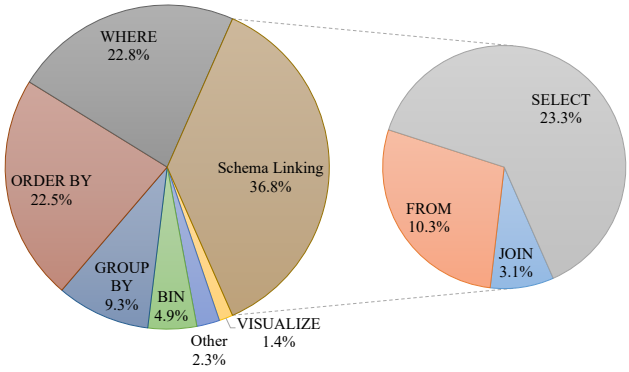

Figure 3: The statistics of the failures of the *NVBench-Feedback* dataset.

Figure 3 reveals the distribution of the inappropriate DVQs in *NVBench-Feedback*, according to the categories of clauses. The number of the inappropriate 'VISUALIZE' clause is the least, whereas that of the clauses related to schema linking is the opposite. The clauses related to schema linking are divided into three categories, the 'SELECT' clause, the 'FROM' clause and the 'JOIN' clause. The 'SELECT' clause is associated with the columns, the 'FROM' clause corresponds to the tables and the JOIN clause is about both. Within the inappropriate clauses related to schema linking, the number of the 'SELECT' clause is larger than the sum of that of the 'FROM' clause and the 'JOIN' clause. The proportion of the number of the 'WHERE' clause is roughly equal to that of the 'ORDER BY' clause, accounting for 22.5%. As for the 'Other' category, it contain multiple invalid DVQs, which include the errors of keywords and those of the syntax.

Additional dataset analysis could be found in Section A.3 of Appendix.

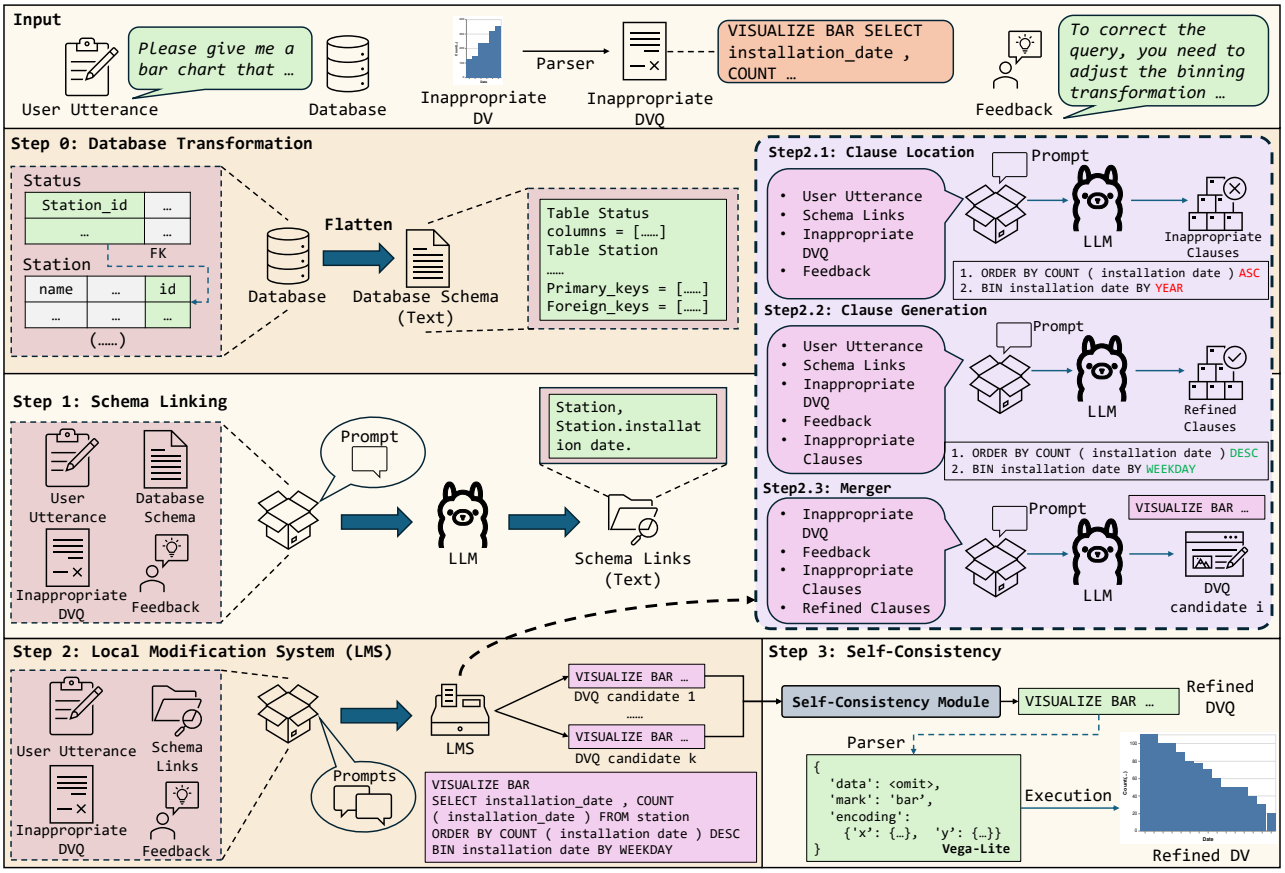

Figure 4: The overview of the *Vis-Edit* framework. The user utterance, the whole database, the inappropriate DVQ and the feedback are passed as input to *Vis-Edit* to refine the DVQ, which will be further used to render into a DV. In Step 0, the whole database is flattened as a sequence of text, referring to the database schema. In Step 1, LLM is utilized to predict the schema links, which is a subset of the database schema. In Step 2, the LMS contains three key modules, the clause location module, the clause generation module and the merger.The clause location module detects the inappropriate clauses of the inappropriate DVQ and the clause generation module refines them. The merger edits the inappropriate DVQ based on the inappropriate clauses and the refined ones, generating one DVQ candidate. Step 1 and Step 2 will be repeated k times to generate k DVQ candidates. In Step 3, the self-consistency module selects the suitable DVQ from the candidates as the result.

4 Methodologies

In this section, the details of the proposed framework, *Vis-Edit*, are ready to be described. Section 4.1 describes the overview of the whole framework, consisting of the Schema Linking module (Section 4.2), the Local Modification System (Section 4.3) and the Self-Consistency module (Section 4.4).

4.1 Overview

The main idea of *Vis-Edit* is to locally edit the inappropriate visualization chart by modifying the corresponding clauses in the DVQ. Most of the inappropriate DVQs, which generated by the immature *Text-to-Vis* model, are closed to the ones that meet the requirement of the user, revealed in Section 3.1. Thus, there is no need to re-generate a new visualization chart based on the natural language feedback. A more efficient method is to re-use the inappropriate DVQs by correcting the inappropriate clauses.

As shown in Figure 4, the user utterance, the whole database, the inappropriate DVQ and the feedback are passed as input to *Vis-Edit* to generate the refined DVQ, which will be further used to render into a DV. *Vis-Edit* contains three key compositions, the schema linking module, the local modification system (LMS) and the self-consistency module. In Step 0, the database is flattened as a sequence of text, referring to the database schema. In Step 1, the database schema, the user utterance, the inappropriate DVQ and the natural language feedback are used to construct the prompt, which will be submitted to LLM. This module analyses the database schema and the user information like the user utterance and the natural language feedback in the prompt to predict the relevant schema links. Such a schema link represents the needed database information required by the task of *Text-to-Vis with Feedback*. For example, the columns used to draw the x-axis and y-axis data in the line chart come from the typical schema links. In Step 2, the LMS consists of the clause location module, the clause generation module and the merger. The clause location module detects the inappropriate clauses and the clause generation module refines the given inappropriate clauses. There are several clauses in each DVQ. Each clause is responsible for rendering a certain part of a visualization chart. For instance, the 'VISUALIZE' clause determines

the type of a chart, mentioned in Section 2.1. The merger module edits the inappropriate DVQ locally, given the information of the inappropriate clauses and refined one. Step 1 and Step 2 will be repeated k times to generated k DV candidates, given the same input. In Step 3, the self-consistency module, it selects the suitable DVQ from the candidates as the final output of *Vis-Edit*.

4.2 Schema Linking

Schema linking (SL) is designed to identify the potential database schema from the whole database, which is a crucial step in data processing pipelines [6, 28, 31]. The schema links refer to these potential database schema, discussed in Section 2.1. Schema linking reduces the search space and improves the performance of model under the cross-domain setting. In the task of *Text-to-Vis with Feedback*, there may be some inconsistency between the database schema and the natural language, making it challenging to refine the inappropriate DVQs directly. For instance, 'allergies' refers to the attribute *Has_Allergy.Allergy*. However, from a computer-based perspective, they are different, especially under the cross-domain setting, where the domain or database in training set differs from that in the test set. The model cannot learn any effective mappings between the natural language and the database schema in the above case. Hence, it is necessary to design a module to bridge the gap.

A prompt-based module is designed for schema linking. Following the *Chain-of-Thought (CoT)* template [43], the LLM is asked to present the thinking process when answering the user questions. It helps produce more accurate and reliable results. *In-context learning (ICL)* is also employed to align the model response with user's intent. Ten-shot approach is integrated into schema linking module.

Specifically, the prompt $\boldsymbol{P}_{sl}$ contains the description I_{sl} of the schema linking task, the user utterance q_i, the whole database schema $\mathcal{D}$, the inappropriate DVQ p_i', the natural language feedback f_i and the random example(s) $\boldsymbol{E}_i = \{e_1, \ldots, e_{|\boldsymbol{E}|}\}$ from the training set. The process of schema linking is formulated as:

$$\mathcal{D}' = LLM[\boldsymbol{P}_{sl}(q_i, \mathcal{D}, p_i', f_i, I_{sl}, \boldsymbol{E}_i)], \tag{2}$$

where $\mathcal{D}' \subseteq \mathcal{D}$ is the schema links. *LLM* is a large language model employed in the framework. Notably, the database schema $\mathcal{D}$ is flattened into text to be embedded in the prompt $\boldsymbol{P}_{sl}$, as shown in Figure 4. The table names, the associated columns, the primary keys and the foreign keys are transformed into the text. A brief example of this module is shown in Section B.1 in the Appendix.

4.3 Local Modification System

Local Modification System (LMS) is composed of clause location module, clause generation module and merger, aiming to make full use of the inappropriate DVQ. In general, the natural language feedback contains many sentences. Each sentence is associated with one or more inappropriate clauses. The following feedback, *To correct the query, you need to adjust the binning transformation to group the installation date by the day of the week instead of by year. Also, sort the count in descending order to show the day of the week with the most installations at the top*, implies that the 'BIN' clause and the order of sorting are inappropriate, respectively. Clause-based method is adopted in the system to detect the inappropriate clauses in a coarse-grained manner and refine them in a fine-grained manner, shown as Figure 4.

4.3.1 Clause Location. The visual elements in a visualization chart corresponds to the clauses of a DVQ. To locate the inappropriate visual elements in a chart based on the natural language feedback, an effective way is to locate the inappropriate clauses. Given a prompt $\boldsymbol{P}_{cl}$ which consists of the description I_{cl} of the task of detecting the clauses, the user utterance q_i, the selected schema links $\mathcal{D}'$, the inappropriate DVQ p_i', the natural language feedback f_i and the random example(s) $\boldsymbol{E}_i$, this module will output the inappropriate clauses $\boldsymbol{W}_i = \{w_1, \ldots, w_{|\boldsymbol{W}_i|}\}$, where w_j is a clause of the inappropriate DVQ p_i'. The specific example is given in Section B.2.1 in the Appendix. The formulation is as follows.

$$\boldsymbol{W}_i = LLM[\boldsymbol{P}_{cl}(q_i, \mathcal{D}', p_i', f_i, I_{cl}, \boldsymbol{E}_i)]. \tag{3}$$

4.3.2 Clause Generation. Clause Generation is to refine the inappropriate clause. Given a prompt $\boldsymbol{P}_{cg}$ which consists of the description I_{cg} of the task of refining the clauses, the user utterance q_i, the selected schema links $\mathcal{D}'$, the inappropriate DVQ p_i', the inappropriate clauses $\boldsymbol{W}_i$ generated by the clause location module, the natural language feedback f_i and the random example(s) $\boldsymbol{E}_i$, this module will output the refined clauses $\boldsymbol{C}_i = \{c_1, \ldots, c_{|\boldsymbol{W}_i|}\}$, where c_j is a refined clause corresponding to the inappropriate clause w_i. The process of this module is formulated as

$$\boldsymbol{C}_i = LLM[\boldsymbol{P}_{cg}(q_i, \mathcal{D}', p_i', f_i, \boldsymbol{W}_i, I_{cg}, \boldsymbol{E}_i)]. \tag{4}$$

Compared with directly editing the whole DVQ, this clause-based method is more accurate, revealed by the experimental results of ablation study in Section 5.2.2. The specific example is given in Section B.2.2 in the Appendix.

4.3.3 Merger. The merger is designed to replace the inappropriate clauses with the refined clauses. Since the clause location module and clause generation module are implemented by LLM, the LLM-generated clauses are probably invalid. For example, the generated inappropriate clauses do not correspond to the original inappropriate DVQ. The synthesized refined clauses may be invalid. Both reasons make it difficult to employ the rule-based method, which simply use existing libraries to replace the given text. To build a robust system, prompt-based merger is proposed. It is easy for LLM to retrieve the similar clauses, though they are not the same. LLM can also implicitly refine these invalid clauses. The specific example is given in Section B.2.3 in the Appendix. Hence, the process is formulated as

$$\boldsymbol{Q}_i = LLM[\boldsymbol{P}_{mg}(p_i', f_i, \boldsymbol{W}_i, \boldsymbol{C}_i, I_{mg}, \boldsymbol{E}_i)], \tag{5}$$

where $\boldsymbol{P}_{mg}$ is the designed prompt template, I_{mg} is the task description and $\boldsymbol{Q}_i$ is the generated DVQ.

4.4 Self-Consistency

The LMS may generate inappropriate DVQ due to the inappropriate schema links generated by the schema linking module. This issue refers to the problem of error propagation. To alleviate this problem, a "sample-and-marginalize" method is employed. All these generated DVQs will be marginalized out to filter out the best answer. Inspired by the idea that diverse perspectives leading to the same conclusion increase confidence in its accuracy, a self-consistency strategy is adopted. The DVQ with the most occurrences will be selected as the final answer. Compared with other re-ranking method,

it avoids any additional training and requires no extra human annotation. Additionally, it reduces the randomness of a single sampled generation, making the editing process more robust. An typical example is given in Section B.3 in the Appendix.

5 Experiment

In this section, the performance of *Vis-Edit* will be evaluated. Section 5.1 introduces the experimental settings. The experimental results are discussed mainly in Section 5.2, including a performance comparison (Section 5.2.1), an ablation study (Section 5.2.2), a parameter study (Section 5.2.3) and a case study (Section 5.2.4). Furthermore, details on the experiment can be found in Section C. The code is publicly available at https://drive.google.com/drive/folders/1fcLTT7FYIZQmWReZfV9d-qelTP4y479E?usp=drive_link.

5.1 Settings

Detailed settings could be found in Section C.1 of Appendix.

5.1.1 Dataset. The diverse dataset *NVBench-Feedback* constructed in Section 3 was used in the evaluation, composed of 48, 482 tuples.

5.1.2 Baselines. Three prompt-based methods, two existing text-to-vis methods and the proposed framework *Vis-Edit* were implemented to conduct the performance comparison. The prompt-based methods are zero-shot, few-shot and the state-of-the-art code generation method, SELF-DEBUGGING [3], which were implemented based on Llama3-8B [5], Mixtral-7B-v0.3 [10] and GPT-4o-mini. The few-shot strategy and SELF-DEBUGGING are under the 10-shot setting. The text-to-vis methods are Seq2Vis [23] and Transformer [39]. As for ncNet [24], it could not be applied to the proposed task, since it does not support the VQL [21, 22], a DVQ employed in the dataset. To ensure the fairness, the input to all the baselines are consistent, that is, including user utterance, inappropriate DVQ, feedback and appropriate DVQ.

5.1.3 Parameters. *Vis-Edit* is evaluated by varying the number of demonstrations n to explore the impact of n to the performance. In this experiment, n is set to 0, 1, 5 and 10. The default value is 1. Following the setting of existing studies [42], the temperature t is set to 0.7. The number of candidates k is set to 10 by default.

5.1.4 Metrics. Vis Accuracy, Data Accuracy, Axis Accuracy, and Overall Accuracy, are employed in this experiment to evaluate the performance, following the settings of the existing studies [23, 37].

5.2 Main Experimental Results

5.2.1 Comparison. *Vis-Edit* is evaluated by being compared with the baseline methods. In Table 4, *Vis-Edit* almost achieves the best performance on various metrics among these baselines, indicating that the proposed framework is effective. Especially, the overall accuracy of *Vis-Edit* surpasses that of the state-of-the-art code generation strategy by 18.58% and 19.46%, augmented with Llama3-8B and GPT-4o-mini, respectively. Notably, the self-consistency module was removed from *Vis-Edit* augmented with GPT-4o-mini (w/o SC) for saving money, which will repeatedly call LLM multiple times. Besides, the existing text-to-vis methods like Seq2Vis [23] and Transformer [39] could not work well in the task, since the overall accuracy of both methods are less than 1%. Inaccurate visualizations from these methods stem from underfitting due to too few trainable parameters. As for ncNet [24], it could not be applied to the proposed task, since it does not support the VQL [21, 22], a DVQ employed in the dataset. Overall, this comparison shows the effectiveness of *Vis-Edit*.

5.2.2 Ablation Study. An ablation study was conducted to explore the contribution of each module. The results are shown in Table 3.

Vis-Edit without the schema linking module (w/o SL) means that the schema linking module is removed from the designed framework. The downstream modules take the whole database schema as input instead of the predicted schema. In *Vis-Edit* without the local modification system (w/o LMS), LLM is asked to directly generated DVQ given the associated schema links, the user utterance, the inappropriate DVQ and the feedback. It employs the Chain-of-Thought strategy and the few-shot setting the same as *Vis-Edit*. As for *Vis-Edit* without the self-consistency module (w/o SC), it refers to that the upstream modules in *Vis-Edit* only generates one DVQ as the final output, equivalent to the greed decoding strategy of LLM. The key observations in this ablation study are as follows.

The schema linking module is still necessary, since the overall accuracy of *Vis-Edit* decreases without this module. Table 3 shows that the schema linking module mainly boosts the overall accuracy of of SELECT, FROM, JOIN and BIN clauses, which are composed of keyword and schema information, thus improving the overall performance by an average of 2.0%. Moreover, the performance improvement of this module increases with the number of demonstrations, indicating that it fully leverages the advantages of *In-Context Learning* to enhance performance without modifying model weights. Therefore, it is crucial for *Vis-Edit* to adopt the schema linking module to predict potential database schema links.

The local modification system (LMS) is the key module of *Vis-Edit*. It contributes the largest growth of the overall accuracy on both general LLMs. With the help of the LMS, the growth in Llama3-8B is greater than 18% and that in Mixtral-7B is greater than 10%, showing that refining the inappropriate clauses is more effective than generating new DVQs from scratch. Furthermore, this module greatly improves the overall accuracy on every clauses, especially for the BIN clause, with a maximum improvement of 46.61%.

The self-consistency module is beneficial to narrow the gap between *Vis-Edit* and the upper bound by improving more than 5% overall accuracy under few-shot setting, where the upper bound refers to the top-10 overall accuracy shown in Table 6. This indicates that this module has the ability to alleviate the problem of error propagation, further building upon the successes of the upstream modules.

Overall, *Vis-Edit* with all modules almost outperforms all other variants with incomplete modules, according to Table 3. This result demonstrates the effectiveness of each module in *Vis-Edit*.

5.2.3 Parameter Study. The respective effects of beam size and the number of demonstrations on performance are explored in this parameter study.

The impact of the beam size on the overall accuracy is evaluated, under the 1-shot setting. The overall accuracy (ACC.@1) increases as the beam size grows, shown in Table 5. This table also reveals the growth rate of the overall accuracy decrease as the beam size grows, which conforms to the law of diminishing marginal utility. Specially, when beam size is larger than 10, the growth increases

Table 3: Ablation Study

Model		Method	Vis Acc.	Axis Acc.		Data Acc.					Acc.
			VISUALIZE	SELECT	FROM	JOIN	WHERE	GROUP BY	ORDER BY	BIN	
Llama3-8B	1-shot	VIS-EDIT	0.9958	**0.9044**	0.9626	0.8600	**0.5997**	**0.9287**	**0.8825**	**0.8263**	**0.7062**
		w/o SL	**0.9960**	0.9006	**0.9628**	**0.8679**	0.5989	0.9245	0.8655	0.7585	0.6954 (-0.0108)
		w/o LMS	0.9291	0.7045	0.8048	0.5779	0.5384	0.8074	0.6339	0.3602	0.4956 (-0.2106)
		w/o SC	0.9946	0.8843	0.9566	0.8372	0.5792	0.9126	0.8492	0.7733	0.6563 (-0.0499)
	5-shot	VIS-EDIT	0.9919	**0.8708**	**0.9306**	**0.8065**	**0.6095**	**0.9221**	**0.8559**	**0.7352**	**0.6844**
		w/o SL	**0.9925**	0.8226	0.9019	0.7601	0.5948	0.9129	0.8246	0.6483	0.6602 (-0.0242)
		w/o LMS	0.8519	0.6307	0.7204	0.4470	0.4820	0.7597	0.6192	0.2924	0.4761 (-0.2083)
		w/o SC	0.9840	0.8291	0.9029	0.7519	0.5678	0.9024	0.8167	0.6886	0.6338 (-0.0506)
	10-shot	VIS-EDIT	0.9942	**0.8807**	**0.9408**	**0.8438**	**0.6340**	**0.9250**	**0.8729**	**0.7500**	**0.7062**
		w/o SL	**0.9952**	0.8448	0.9160	0.7879	0.6201	0.9176	0.8421	0.7055	0.6769 (-0.0293)
		w/o LMS	0.8546	0.6893	0.7303	0.4619	0.4788	0.7782	0.6545	0.4237	0.5183 (-0.1879)
		w/o SC	0.9900	0.8513	0.9112	0.7596	0.6005	0.8939	0.8285	0.6822	0.6423 (-0.0639)
Mixtral-7B	1-shot	VIS-EDIT	0.9935	**0.8729**	0.9628	0.8795	**0.5237**	**0.8487**	**0.8277**	0.6928	**0.6051**
		w/o SL	**0.9942**	0.8638	**0.9632**	**0.8844**	0.5172	0.8468	0.8175	**0.7034**	0.5999 (-0.0052)
		w/o LMS	0.8702	0.7115	0.7767	0.5613	0.5082	0.7747	0.6280	0.3263	0.4769 (-0.1282)
		w/o SC	0.9913	0.8352	0.9511	0.8592	0.4722	0.8211	0.7879	0.6419	0.5503 (-0.0548)
	5-shot	VIS-EDIT	**0.9881**	**0.8682**	**0.9431**	0.8563	**0.5547**	**0.8932**	**0.8274**	**0.6864**	**0.6490**
		w/o SL	**0.9881**	0.8513	0.9429	**0.8575**	0.5409	0.8884	0.8020	0.6674	0.6226 (-0.0264)
		w/o LMS	0.8978	0.6954	0.8316	0.6653	0.4935	0.7787	0.6782	0.3263	0.5102 (-0.1388)
		w/o SC	0.9850	0.8302	0.9279	0.8277	0.5139	0.8734	0.7751	0.6165	0.5907 (-0.0583)
	10-shot	VIS-EDIT	**0.9890**	**0.8907**	**0.9545**	**0.8906**	**0.5523**	**0.8895**	**0.8232**	0.7733	**0.6673**
		w/o SL	0.9869	0.8801	0.9495	0.8757	0.5409	0.8768	0.8006	**0.7903**	0.6438 (-0.0235)
		w/o LMS	0.9209	0.7794	0.8832	0.7680	0.5449	0.8074	0.7102	0.4576	0.5622 (-0.1051)
		w/o SC	0.9854	0.8486	0.9365	0.8513	0.5180	0.8563	0.7797	0.7119	0.6018 (-0.0655)

Table 4: Main Performance Comparison.

Model	Method	Vis Acc.	Axis Acc.	Data Acc.	Acc.
Mixtral-7B	Zero-Shot	0.9646	0.8064	0.7935	0.4646
	Few-Shot	0.9823	0.8098	0.8107	0.4665
	Self-Debugging	0.9840	0.8771	**0.8872**	0.5955
	Vis-Edit	**0.9890**	**0.9226**	0.8489	**0.6673**
Llama3-8B	Zero-Shot	0.9725	0.6943	0.7375	0.3849
	Few-Shot	0.9720	0.8052	0.7879	0.4911
	Self-Debugging	0.9771	0.8352	0.8382	0.5204
	Vis-Edit	**0.9942**	**0.9108**	**0.8729**	**0.7062**
GPT-4o-mini	Zero-Shot	0.9642	0.8374	0.8389	0.5208
	Few-Shot	0.9906	0.8667	0.8420	0.5664
	Self-Debugging	0.9940	0.8828	0.8641	0.5836
	Vis-Edit	**0.9942**	**0.9538**	**0.9178**	**0.7782**
Text-to-Vis	Seq2Vis	<0.010	<0.010	<0.010	<0.010
	Transformer	<0.010	<0.010	<0.010	<0.010
	ncNet	-	-	-	-

Table 5: The overall accuracy varying different beam size

	k=1	k=5	k=10	k=15	k=20	k=40
Llama3-8B	0.6396	0.6877	0.7062	0.7106	0.7151	0.7179
Mixtral-7B	0.5437	0.5920	0.6051	0.6132	0.6161	0.6182

Table 6: The overall accuracy varying different n.

Model	Metric	n=0	n=1	n=5	n=10
Mixtral-7B	Acc.@1	0.5137	0.6051	0.6490	0.6673
	Acc.@10	0.7001	0.7514	0.7628	0.7807
Llama3-8B	Acc.@1	0.5024	0.7062	0.6844	0.7062
	Acc.@10	0.6833	0.8017	0.7836	0.8165

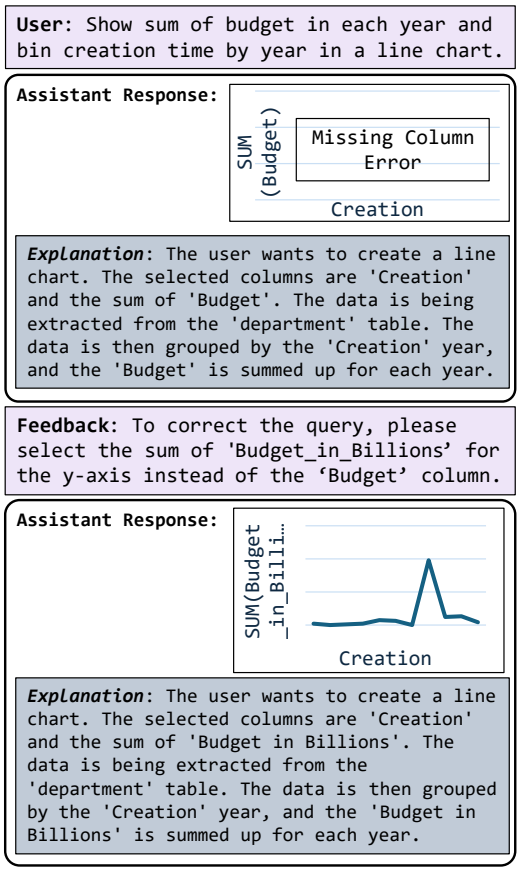

Figure 5: Case Study with the feedback in technical language

slowly with the increase of the beam size. The growth of performance with majority voting validates the hypothesis in Section 4.4 that diverse reasoning paths will lead to the same conclusion to increase confidence in the performance of model. Moreover, these results indicate not only that a larger beam size will yield better performance, but also that a proper beam size, $k = 10$ in this setting, can be chosen to achieve the trade-off between the performance and the cost. Besides, the law of diminishing marginal utility suggests that the general LLMs is the bottleneck of the overall performance.

The effects of the number of demonstrations on the performance of *Vis-Edit* under different LLMs are shown in Table 6. Given the fix beam size, $k = 10$, the top-1 and top-10 overall accuracy is studied varying different number of demonstrations. Both metrics generally increase as the number of demonstrations grows for the two LLMs. The rising top-10 overall accuracy suggests that the LLMs can generate DVQ candidates more accurately with additional demonstrations. Similarly, the increasing top-1 overall accuracy indicates that the LLMs can produce more correct DVQs as the number of demonstrations increases. As for the minor drop which occurs when the number of demonstrations is 5 based on Llama3-8B, this is probably because of the random factors like the internal parameters of the general LLM.

Overall, this parameter study shows that the performance of *Vis-Edit* improves with increases in beam size or the number of demonstrations. This result confirms that these two parameters are beneficial factors for *Vis-Edit*. Supplemental results of parameter study could be found in Section C.2.1 of Appendix.

5.2.4 Case Study. The case study was conducted to test whether the *Vis-Edit* can provide services for the users from different backgrounds. As shown in Figure 1, the user is unfamiliar with the technical tools in database schema or visualization. It only asked the system to correct the visualization in layman language. *Vis-Edit* then satisfied the intent of the user successfully. In Figure 5, the user has the background knowledge of the database schema. After being queried, the *Text2Vis* model visualized the 'Budget' column which does not exist in the database, since the information of user utterance is not complete. The correct one should be 'Budget_in_Billions'. With the help of the explanation of how the inappropriate visualization chart was constructed, the user could provide the technical feedback to specify the exact column to visualize. *Vis-Edit* could handle this case smoothly.

Overall, the case study demonstrates the effectiveness of *Vis-Edit* on the layman language feedback and the technical feedback.

6 Related Work

This study is closely related to the field of the Text-to-Vis, the LLM for data visualization and the interactive systems with user engagement, which will be briefly discussed.

Text-to-Vis. *Text-to-Vis* has great importance to many practical applications. Early Text-to-Vis system employed rule-based methods. Articulate [38], DataTone [7] and Eviza [33] utilized symbolic NLP methodologies to analyse the natural language and convert it to data visualizations. NL4DV [27] and FlowSense [46] integrated powerful semantic parsers [18, 26] to further improve the performance of the system. Although these rule-based methods are easy to implement, they could not comprehend complex natural language. Neural methods [23, 24, 36, 37] were used to handle the complex natural language. Specifically, Luo et al. trained ncNet [24], an encoder-decoder architectural network, for the benchmark [23] of the task. RGVisNet [37] adopted the retrieval and generation methodology to generate the data visualizations, motivated by the process of code generation. MMCoVisNet [36] studied the scenario of users interacting with the system. The core objective of each dialogue session is to develop an appropriate data visualization.

However, most of the existing studies ignored the interaction between the user and the text-to-vis system. The user can only re-generate a data visualization instead of instructing the system to edit it, if the old one could not meet its requirements. Compared with these existing text-to-vis studies, *Vis-Edit* further considers the effect of natural language feedback, enhancing the capabilities of the existing text-to-vis system. Although this study [36] investigated user interaction with the DV system, it failed to leverage user feedback to refine inappropriate DVQs. Additionally, the associated dataset lacks feedback-integrated data for DVQ refinement.

LLM for Data Visualization. It is vital to leverage LLMs to realize the tasks related to data visualization, because of the remarkable capability of comprehending natural languages and generating high-quality responses in user-defined formats [16]. Chat2VIS [25] visualizes data by generating Python code with LLM prompt. LIDA [4] modeled the Text-to-Vis task as a four-stage generation problem, combining the LLMs with the image generation models to synthesize the data visualizations and the infographics. MatPlotAgent [45] designed a multi-agent framework to address the scientific data visualization tasks. It contains a query understanding module, code generation module with iterative debugging, and a visual feedback mechanism for error correction. ChartMimic [34] is a multi-modal code generation model, developed to measure the proficiency capability of LLMs via chart-to-code generation.

However, Chat2VIS, LIDA and MatPlotAgent focused on the Text-to-Vis problem, while *Vis-Edit* addresses the task of Text-to-Vis with feedback. Although MatPlotAgent employs the visual feedback to enhance the performance, the scenarios of MatPlotAgent and *Vis-Edit* are different. The feedback of MatPlotAgent is from the visual element analysis of the visual model, while the feedback in *Vis-Edit* is based on the natural language explanation of generated DVQ. Moreover, it does not consider the problem of schema linking since the data to visualize is given. In *Vis-Edit*, the data to visualize should be selected from the whole database, which is more challenging.

Interactive Systems with User Engagement. Interactive systems have garnered significant attention due to their importance in a large number of applications. In software engineering, the workflow of ITDCG [11] makes full use of the lightweight user feedback to generate tests and code suggestions, which satisfy the user intent. The study [2] proposed imitation learning from language feedback (ILF) for code generation, making the training process user-friendly and sample-efficient. In computer vision, ChatIR [13] engages in an interaction with the user to specify the search intent of the user, improving the performance of image retrieval. TiGAN [48] utilized natural language feedback to guide image generation and manipulation, lowering the barriers of image editing. In recommendation system, the research [44] studied the feasibility of the multi-modal

conversational recommendation models with both positive and negative natural-language feedback.

However, the existing interactive systems do not study the field of data visualization. Hence, there is a need to further explore the feasibility of the interactive systems of data visualization. To the best of our knowledge, *Vis-Edit* is the first study focusing on the interactive system of data visualization, promoting the development of the interactive systems based on the user engagement.

7 Conclusion and Future Work

In this paper, a new task named *Text-to-Vis with feedback* is proposed to enhance the interactive capabilities of a Text-to-Vis system. This paper is the first work to explore this interactive system. To further study the proposed task, a new dataset *NVBench-Feedback* was constructed. A LLM-based framework called *Vis-Edit* is also proposed. It focuses on re-using the inappropriate predictions through refining the inappropriate clauses, proven to be more effective than the direct editing on the queries. The schema linking module and the self-consistency were integrated into the framework to alleviate two issues, the inconsistency between the database schema and the natural language and the problem of error propagation. Comparison among baselines and ablation study valid the effeteness of the framework. In all cases, the accuracy of *Vis-Edit* is better than all baselines by at least 7.18% and up to 19.46%. The parameter study and the case study also give the insights to take the advantage of the framework.

Building on this line of research, further investigation into additional scenarios, such as realistic datasets with user interactions and handling a new domain without schema information, is warranted. Moreover, while the self-consistency module has shown effectiveness in selecting optimal candidates, it is plagued by efficiency issues. In this regard, exploring more sophisticated designs for a more efficient re-ranking module could drive further improvements.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Angelica Chen, Jérémy Scheurer, Jon Ander Campos, Tomasz Korbak, Jun Shern Chan, Samuel R. Bowman, Kyunghyun Cho, and Ethan Perez. 2024. Learning from Natural Language Feedback. *Transactions on Machine Learning Research* (2024).

[3] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching Large Language Models to Self-Debug. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=KuPixIqPiq

[4] Victor Dibia. 2023. LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics using Large Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. 113–126.

[5] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[6] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (Jan. 2024), 1132–1145. doi:10.14778/3641204.3641221

[7] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G Karahalios. 2015. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of the 28th annual acm symposium on user interface software & technology*. 489–500.

[8] Pat Hanrahan. 2006. Vizql: a language for query, analysis and visualization. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. 721–721.

[9] Valentina Ivančić. 2014. Improving the decision making process trought the Pareto principle application. *Ekonomska misao i praksa* 23, 2 (2014), 633–656.

[10] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[11] Shuvendu K Lahiri, Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, Madanlal Musuvathi, Piali Choudhury, Curtis von Veh, Jeevana Priya Inala, Chenglong Wang, et al. 2022. Interactive code generation via test-driven user-intent formalization. *arXiv preprint arXiv:2208.05950* (2022).

[12] VI Lcvenshtcin. 1966. Binary coors capable or 'correcting deletions, insertions, and reversals. In *Soviet Physics-Doklady*, Vol. 10.

[13] Matan Levy, Rami Ben-Ari, Nir Darshan, and Dani Lischinski. 2024. Chatting makes perfect: Chat-based image retrieval. *Advances in Neural Information Processing Systems* 36 (2024).

[14] Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025. Alpha-sql: Zero-shot text-to-sql using monte carlo tree search. *arXiv preprint arXiv:2502.17248* (2025).

[15] Deqing Li, Honghui Mei, Yi Shen, Shuang Su, Wenli Zhang, Junting Wang, Ming Zu, and Wei Chen. 2018. ECharts: a declarative framework for rapid construction of web-based visualization. *Visual Informatics* 2, 2 (2018), 136–146.

[16] Guozheng Li, Xinyu Wang, Gerile Aodeng, Shunyuan Zheng, Yu Zhang, Chuangxin Ou, Song Wang, and Chi Harold Liu. 2024. Visualization Generation with Large Language Models: An Evaluation. *arXiv preprint arXiv:2401.11255* (2024).

[17] Shuaimin Li, Xuanang Chen, Yuanfeng Song, Yunze Song, Chen Jason Zhang, Fei Hao, and Lei Chen. 2025. prompt4vis: prompting large language models with example mining for tabular data visualization. *The VLDB Journal* 34, 4 (2025), 1–26.

[18] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*. 63–70.

[19] Yuyu Luo, Chengliang Chai, Xuedi Qin, Nan Tang, and Guoliang Li. 2020. Visclean: Interactive cleaning for progressive visualization. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2821–2824.

[20] Yuyu Luo, Xuedi Qin, Chengliang Chai, Nan Tang, Guoliang Li, and Wenbo Li. 2020. Steerable self-driving data visualization. *IEEE Transactions on Knowledge and Data Engineering* 34, 1 (2020), 475–490.

[21] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018. Deepeye: Towards automatic data visualization. In *2018 IEEE 34th international conference on data engineering (ICDE)*. IEEE, 101–112.

[22] Yuyu Luo, Xuedi Qin, Nan Tang, Guoliang Li, and Xinran Wang. 2018. Deepeye: Creating good data visualizations by keyword search. In *Proceedings of the 2018 International Conference on Management of Data*. 1733–1736.

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

[24] Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2021. Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics* 28, 1 (2021), 217–226.

[25] Paula Maddigan and Teo Susnjak. 2023. Chat2VIS: generating data visualizations via natural language using ChatGPT, codex and GPT-3 large language models. *Ieee Access* 11 (2023), 45181–45193.

[26] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 55–60.

[27] Arpit Narechania, Arjun Srinivasan, and John Stasko. 2020. NL4DV: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2020), 369–379.

[28] Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems* 36 (2023), 36339–36348.

[29] Thomas Pyzdek. 2021. Pareto Analysis. In *The Lean Healthcare Handbook: A Complete Guide to Creating Healthcare Workplaces*. Springer, 157–164.

[30] Xin Qian, Ryan A Rossi, Fan Du, Sungchul Kim, Eunyee Koh, Sana Malik, Tak Yeon Lee, and Joel Chan. 2021. Learning to recommend visualizations from data. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 1359–1369.

[31] Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X Sean Wang. 2024. Purple: Making a large language model a better sql writer. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 15–28.

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

[33] Vidya Setlur, Sarah E Battersby, Melanie Tory, Rich Gossweiler, and Angel X Chang. 2016. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th annual symposium on user interface software and technology*. 365–377.

[34] Chufan Shi, Cheng Yang, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran Xu, Xinyu Zhu, Siheng Li, Yuxiang Zhang, et al. 2024. ChartMimic: Evaluating LMM's Cross-Modal Reasoning Capability via Chart-to-Code Generation. *arXiv preprint arXiv:2406.09961* (2024).

[35] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya Parameswaran. 2016. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *Proceedings of the VLDB Endowment* 10, 4 (2016), 457–468.

[36] Yuanfeng Song, Xuefang Zhao, and Raymond Chi-Wing Wong. 2024. Marrying dialogue systems with data visualization: Interactive data visualization generation from natural language conversations. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2733–2744.

[37] Yuanfeng Song, Xuefang Zhao, Raymond Chi-Wing Wong, and Di Jiang. 2022. Rgvisnet: A hybrid retrieval-generation neural framework towards automatic data visualization generation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1646–1655.

[38] Yiwen Sun, Jason Leigh, Andrew Johnson, and Sangyoon Lee. 2010. Articulate: A semi-automated model for translating natural language queries into meaningful visualizations. In *Smart Graphics: 10th International Symposium on Smart Graphics, Banff, Canada, June 24-26, 2010 Proceedings 10*. Springer, 184–195.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[40] Randle Aaron M Villanueva and Zhuo Job Chen. 2019. ggplot2: elegant graphics for data analysis.

[41] Zhuoyue Wan, Yuanfeng Song, Shuaimin Li, Chen Jason Zhang, and Raymond Chi-Wing Wong. 2025. DataVisT5: A Pre-Trained Language Model for Jointly Understanding Text and Data Visualization . In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1704–1717. doi:10.1109/ICDE65448.2025.00131

[42] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=1PL1NIMMrw

[43] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.

[44] Yaxiong Wu, Craig Macdonald, and Iadh Ounis. 2022. Multimodal conversational fashion recommendation with positive and negative natural-language feedback. In *Proceedings of the 4th Conference on Conversational User Interfaces*. 1–10.

[45] Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, et al. 2024. Matplotagent: Method and evaluation for llm-based agentic scientific data visualization. *arXiv preprint arXiv:2402.11453* (2024).

[46] Bowen Yu and Cláudio T Silva. 2019. FlowSense: A natural language interface for visual data exploration within a dataflow system. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 1–11.

[47] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

[48] Yufan Zhou, Ruiyi Zhang, Jiuxiang Gu, Chris Tensmeyer, Tong Yu, Changyou Chen, Jinhui Xu, and Tong Sun. 2022. Tigan: Text-based interactive image generation and manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 3580–3588.

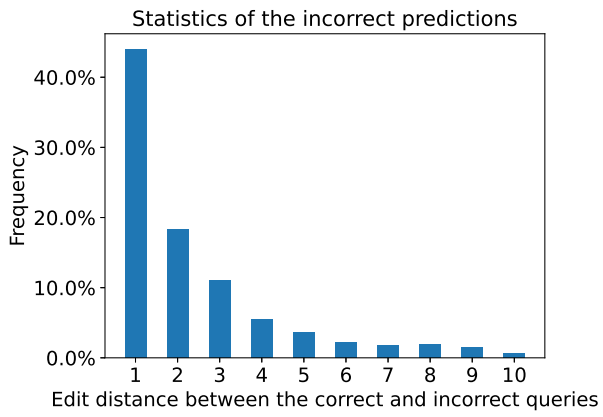

Figure 6: The frequency histogram varying different values of edit distance

A Dataset Construction

In this section, the approach for constructing the dataset for our proposed task, *Text-to-Vis with Feedback*, will be described in detail. The inappropriate data visualization queries are generated (Section A.1), employing the existing *Text-to-Vis* model. These inappropriate queries will be used to synthesize the associated nature language feedback, with the help of the powerful LLMs (Section A.2). These feedback will be further analyzed in Section A.3.

A.1 Generating the Inappropriate Data Visualization Query

The public *NVBench* [23] dataset is used as our source of utterances. It contains 25,750 pairs of natural language and the corresponding DVQs. The DVQs are from diverse databases, i.e., 153 databases, since it is modified based on *Spider* [47], a dataset used for Text-to-SQL where the system generates the SQL given the input text. The incorrect predictions from a text-to-vis model, e.g., Seq2Vis [23], were selected to ensure that our dataset reflects the distribution of errors made by the existing model. The task of generating the inappropriate DVQ is defined as: given a utterance q and the associated database schema $\boldsymbol{D}$, the text-to-vis model generates the DVQ. The DVQs that are inconsistent with the ground-truth are selected as the inappropriate DVQ set.

To increase the number of the elements in the inappropriate DVQ set, the whole *NVBench* dataset was randomly divided into five equal parts. One part of the dataset was used to test the text-to-vis model, while the other parts were used for training. This training and evaluation process was repeated five times, with a different part of the dataset being used as the test set each time. Besides, a beam search of size 3 was used to make the predictions. Modifying the beam width to generate multiple queries is a reasonable approach since the output of the beam search represents the possible predictions made by the model. Therefore, inaccurate DVQs among these predictions could serve as the representations of the errors made by the model in practical scenarios.

To explore the relationship between the inappropriate DVQs and the refined ones, the edit distance was employed for all cases where the text-to-vis model made an inappropriate prediction on the test set. The edit distance is measured by the minimum number of the token-level edit operations required to transform the inappropriate visualization query to the refined one. The edit operations are insertion, deletion and replacement. Figure 6 shows a frequency histogram varying different values of edit distance, based on the result of the *Levenshtein distance algorithm* [12], which is an algorithm for computing the edit distance. The frequency histogram reflects that the edit distance of most inappropriate predictions (80.0%) is within 4, following the *long-tail* distribution. The frequency of those query with edit distance larger than 10 are omitted in the figure. In the *NVBench-Feedback* dataset, those DVQs with long edit distance (i.e., greater or equal to 5) are removed, according to the *Pareto Analysis* [29]. This method of analysis indicates that the 80% of the problems are due to the 20% of the causes, where sharpening the focus to the critical few sources that are causing most of the problems is beneficial to improve a process. In the task of *Text-to-Vis with Feedback*, the DVQs with an edit distance less than 4 are considered as the critical data that contributes significantly to the task, since these DVQs account for nearly 80% of the whole inappropriate DVQ set. It is also similar to [9].

A.2 Generating the Associated Feedback

Large language model (LLM), like Llama3-70B[5], was used to synthesize the natural language feedback, since it has a lot of advantages. For example, a large number of feedback data is generated at low cost with the help of the powerful LLMs. Compared with utilizing the rule-based methods, the synthetic data of LLMs is rich in diversity. The framework contains two parts, explanation generator and feedback generator, shown in Figure 7. The explanation generator explains the inappropriate DVQ and the refined one in a human-understandable way. The feedback generator provides feedback to modify the inappropriate DVQ, given the user utterance, both queries and the corresponding explanations. In addition, some constrains of the output format (e.g., JSON format) were added into the prompt to guarantee the quality of feedback.

Here is an example that illustrates how the pipeline synthesizes the samples. Given the inappropriate DVQ *Visualize BAR SELECT Allergy , count(*) FROM Allergy_Type GROUP BY Allergy* and the refined one *Visualize BAR SELECT Allergy , count(*) FROM Has_allergy GROUP BY Allergy*, the Explanation Generator generates the corresponding explanations for both queries, respectively. Both explanations are shown in Figure 1. Then, given the user utterance, the inappropriate query, the refined query and their corresponding explanations, the Feedback Generator generates the feedback, *I want to visualize the specific allergies instead of the general allergy types* in JSON format, based on the prompt shown in Figure 7.

The size of *NVBench-Feedback* is 48,482. It is split under the cross-domain setting. That is, the specific domains (e.g., restaurants and flights) of each split are different. In addition, human evaluation is employed to ensure the quality of the synthesized feedback. Specifically, the accuracy of the test set feedback is about 92.53%.

A.3 Dataset Analysis

The distribution of the visualization types and the edit distance of the DVQs are revealed in Figure 8. The number of the DVQs of bar

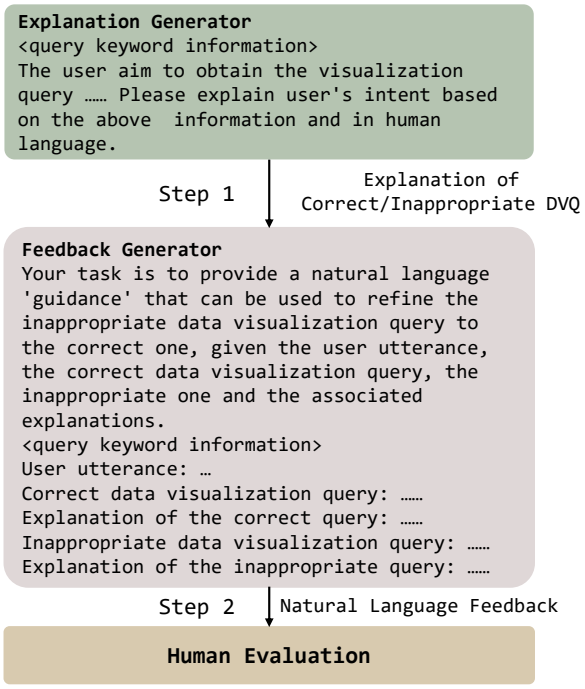

Figure 7: The pipeline of the dataset construction. In Step 1, the explanation generator takes the inappropriate DVQ and the refined one to generate the corresponding natural language explanation, respectively. The query keyword information refers to Section 2.1. In Step 2, the feedback generator employs the given prompt the synthesize the feedback.

chart dominates, and that of the grouping line chart is the least. In addition, most of the edit distance between the inappropriate DVQ and the refined one is 1, while the least is 4.

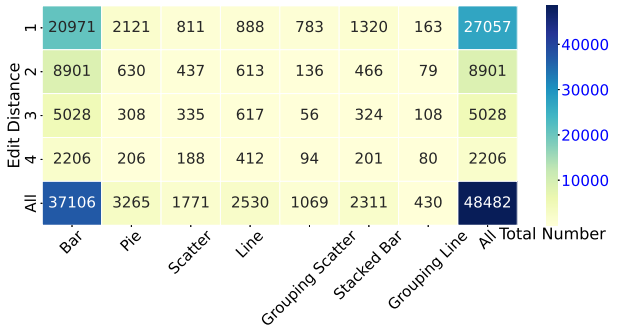

Figure 8: The statistics of the *NVBench-Feedback* dataset.

B Methodologies

In this section, the running examples of the workflow of each module are introduced.

B.1 Schema Linking

Schema linking (SL) is designed to identify the potential database schema from the whole database. A brief example of this module is shown as Figure 9. A prompt contains a database schema, a natural language utterance, an inappropriate DVQ, the corresponding feedback, the task description and the examples. The task description is a text that instructs the LLM to predict the schema links and the examples are the tuples that including the input and the output of this module. As shown in Figure 9, the schema linking module outputs a text response that contains the reason and the answer. Specifically, this module selected $Set\ 1 = [station, station.installation\_date]$ as the schema links, based on the utterance. Besides, the schema in the inappropriate prediction is also chosen to form the schema links, based on the assumption that the schema in the inappropriate DVQ may be satisfying. Hence, $Set\ 2 = [station, station.installation\_date]$. In addition, $Set\ 3 = [station]$ is derived from the natural language feedback. The final output is the union of the three sets, $Schema\ Links = [station, station.installation\_date]$. This generation is based on the *Chain-of-Thought* strategy.

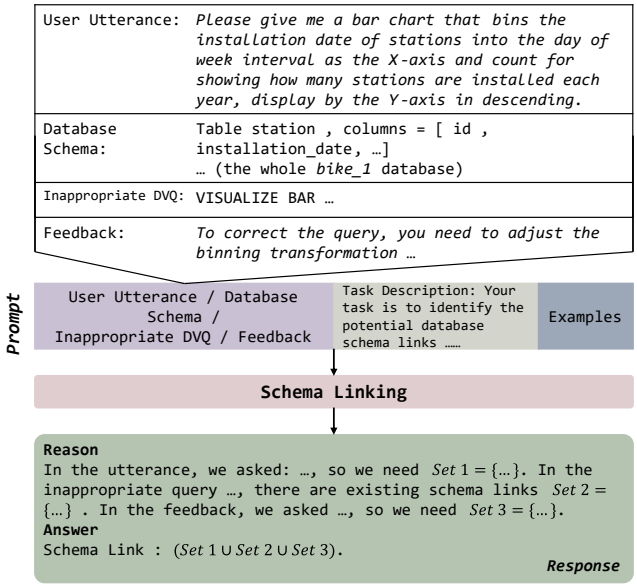

Figure 9: The schema linking module.

B.2 Local Modification System

Local Modification System (LMS) is composed of clause location module, clause generation module and merger, aiming to make full use of the inappropriate DVQ.

B.2.1 Clause Location. Clause Location is to locate the inappropriate clause of the given DVQ. As shown in Figure 4, the clause location module takes the prompt, which contains the user utterance, the inappropriate DVQ, the schema links and the feedback, as input. This prompt also includes the task description similar to that of the schema linking module. The module outputs the inappropriate clauses, the 'ORDER BY' clause and the 'BIN' clause.

B.2.2 Clause Generation. Clause Generation is to refine the inappropriate clause. Here is an example. As shown in Figure 4, the clause generation module takes the prompt, which contains the user utterance, the inappropriate DVQ, the inappropriate clauses (the 'ORDER BY' clause and the 'BIN' clause), the schema links and the feedback, as input. This prompt also includes the task description similar to that of the schema linking module. The module output the refined 'ORDER BY' clause and the refined 'BIN' clause.

B.2.3 Merger. The merger is designed to replace the inappropriate clauses with the refined clauses. Here is an example. As shown in Figure 4, given the inappropriate DVQ, the inappropriate 'ORDER BY' clause, the inappropriate 'BIN' clause and the refined ones, the merger output the refined DVQ candidate, as one of the input of the following self-consistency module.

B.3 Self-Consistency

Figure 10 is employed as an example. Suppose that k is equal to 10, indicating that the schema linking module and the LMS are repeated 10 times, generating 10 DVQ candidates. Within these candidates, the number of the 'DVQ 1' is 3 and that of the 'DVQ 2' is 7. 'DVQ 2' is selected as the final answer since it occurs with the highest frequency.

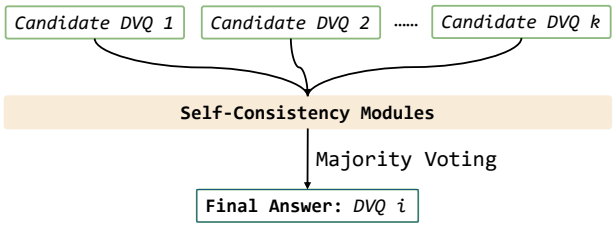

Figure 10: The self-consistency module.

C Experiment

In this section, supplementary details of the experiment are introduced.

C.1 Settings

The experiments were conducted based on Pytorch 2.2.0 and CUDA 11.8, on a server running Almalinux 9 OS with a 24-core CPU, eight NVIDIA RTX3090 GPU and 128.0GB RAM.

C.1.1 Baselines. Three prompt-based methods, two existing text-to-vis methods and the proposed framework *Vis-Edit* were implemented to conduct the performance comparison. The prompt-based methods are zero-shot, few-shot and the state-of-the-art code generation method, SELF-DEBUGGING [3], which were implemented based on Llama3-8B [5], Mixtral-7B-v0.3 [10] and GPT-4o-mini. The few-shot strategy and SELF-DEBUGGING are under the 10-shot setting. The text-to-vis methods are Seq2Vis [23] and Transformer [39]. As for ncNet [24], it could not be applied to the proposed task, since it does not support the VQL [21, 22], a DVQ employed in the dataset. To ensure the fairness, the input to all the baselines are consistent, that is, including user utterance, inappropriate DVQ, feedback and appropriate DVQ.

- Zero-Shot and Few-Shot. They are the general LLM prompt strategies, utilizing the Chain-of-Thought strategy [43].
- Self-Debugging. It is the state-of-the-art code generation method, proposed by [3]. However, since the unit tests are not available in the task of *Text-to-Vis with Feedback*, Self-Debugging should be modified to adapt this task. Specifically, the generated code in Self-Debugging was replaced with the inappropriate DVQ and the explanation feedback was replaced with that synthesized in Section A. Besides, it was implemented as the one-round conversation problem, consistent with *Vis-Edit*.
- Text-to-vis methods. Since these text-to-vis methods could not directly take the natural language feedback as input, they were modified to take the text sequence as input, which were concatenated by the user utterance, database schema, the inappropriate DVQ and the natural language feedback.

C.1.2 Evaluation Metrics.

- Overall Accuracy. It directly compares the predicted DVQ and the ground-truth DVQ token by token. The accuracy is calculated as $Acc. = N_{matched}/N_{total}$, where $N_{matched}$ is the number of the matched DVQs and N_{total} is the number of the DVQs being evaluated. Compared with other metrics, it reflects the comprehensive performance of the models.
- Vis Accuracy. Each DVQ contains three components, visualization type, x/y/z-axis, and data transformations. Vis accuracy measures the matches of the visualization types components between the generated DVQ and the ground-truth DVQ. It is defined as $Vis\ Acc. = N_{vis}/N_{total}$, where N_{vis} is the number of visualization type components matching the ground-truth result and N_{total} is the number of the DVQs being evaluated.
- Data Accuracy. This measurement reflects the matches of the data transformation components between the generated DVQ and the ground-truth DVQ. It is defined as $Data\ Acc. = N_{data}/N$, where N_{data} is the number of data transformation components matching the ground-truth result and N is the number of the data transformation components being evaluated.
- Axis Accuracy. It calculates the matches of the x/y/z-axis components between the generated DVQ and the ground-truth DVQ. The measurement is defined as $Axis\ Acc. = N_{axis}/N'$, where N_{axis} is the number of x/y/z-axis components matching the ground-truth result and N' is the number of the x/y/z-axis components being evaluated.

C.2 Supplementary Experimental Results

C.2.1 Parameter Study. The chart of the overall accuracy varying different beam size is shown as follows.

C.2.2 Efficiency Analysis. We conducted an efficiency analysis experiment using one NVIDIA RTX 3090 GPU and 128.0GB of RAM, with Llama3-8B-Instruct deployed for the tasks. We evaluated the running time of Vis-Edit under varying numbers of candidates, denoted by K. For this analysis, the entire test set with 5,194 items was utilized. The results are as follows.

- R1: The running time of Vis-Edit with K=10 is ten times that with K=1.
- R2: The running time of Vis-Edit with K=20 is not twice that with K=10.
- R3: The running time of Vis-Edit with K=40 is not four times that with K=10.

K	1	10	20	40
Time (s/item)	19.91	197.94	264.29	439.90

Table 7: The running time of Vis-Edit varying different values of K

The first experimental result (R1) is consistent with intuition, since the number of candidates generated by Vis-Edit with K=10 is ten times that with K=1.

The results of the second and third experiments (R2 and R3) are not intuitive. This is because we conduct temperature sampling on the LLM and the temperature is set to 0.7, following the setting of existing studies [36]. During the generation of each candidate, the intermediate results are different, which implies that the number of tokens consumed during the generation of each item are different. The number of tokens consumed of Vis-Edit with K=20 may be not twice that with K=10, so the running time of Vis-Edit with K=20 is not twice that with K=10.

To reduce the runtime of Vis-Edit during deployment, parallelization can be leveraged, as the generation of each candidate query is independent of others. For instance, multiple processes can be used to call the OpenAI API (e.g., GPT-4o-mini) in parallel. With the GPT-4o-mini API and K set to 10, the average runtime for refining a single DVQ is 9.71 seconds.

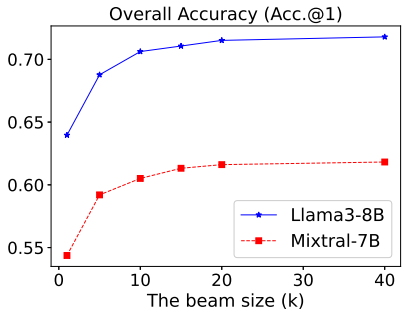

Figure 11: The overall accuracy varying different beam size.

C.3 Summary

This section evaluates the performance of the proposed framework, *Vis-Edit*, through a comprehensive comparison with baseline methods, an ablation study, a parameter study, and a case study. The general comparison reveals that *Vis-Edit* significantly outperforms the existing baselines. In all cases, the accuracy of *Vis-Edit* is better than all baselines by 7.18%. The ablation study demonstrates the effectiveness of each internal module by presenting the exact match accuracy for each clause under various settings. The parameter study investigates the impact of two factors, beam size and the number of demonstrations, finding that both factors positively influence the framework. The case study shows that *Vis-Edit* can accommodate users with varying levels of background knowledge. Overall, these experiments confirm the viability of *Vis-Edit*, and the associated analysis provides valuable guidance for users to effectively utilize the framework.