SQLucid: Grounding Natural Language Database Queries with Interactive Explanations

Yuan Tian
Purdue University
West Lafayette, IN, USA
tian211@purdue.edu

Jonathan K. Kummerfeld
University of Sydney
Sydney, Australia
jonathan.kummerfeld@sydney.edu.au

Toby Jia-Jun Li
University of Notre Dame
Notre Dame, IN, USA
toby.j.li@nd.edu

Tianyi Zhang
Purdue University
West Lafayette, IN, USA
tianyi@purdue.edu

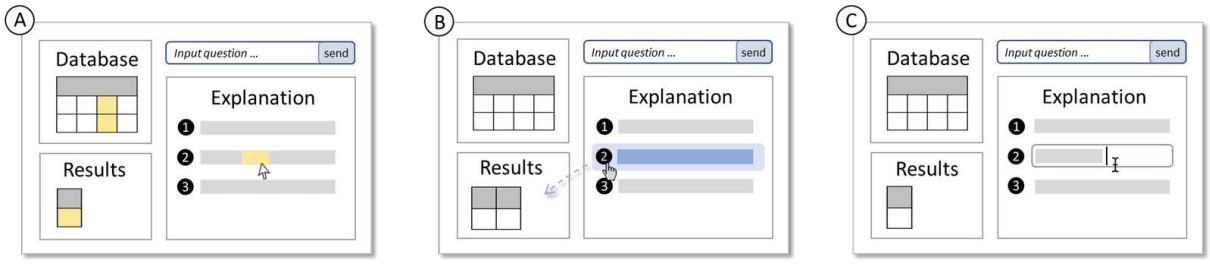

Figure 1: An Overview of SQLucid: Ⓐ Given a model-generated SQL query, SQLucid helps the user understand the query behavior by generating a step-by-step explanation in natural language. When the user hovers over an entity (e.g., a column name) in the natural language explanation, SQLucid will highlight the corresponding elements in the database and query result to help the user grasp the visual correspondence between the explanation and the database. Ⓑ For each step in the natural language explanation, the user can inspect the intermediate query result of the step to validate query behavior and diagnose query errors. Ⓒ Once the user identifies the erroneous step, they can directly edit the explanation of that step to specify the correct behavior and guide the model to refine the query.

ABSTRACT

Though recent advances in machine learning have led to significant improvements in natural language interfaces for databases, the accuracy and reliability of these systems remain limited, especially in high-stakes domains. This paper introduces SQLucid, a novel user interface that bridges the gap between non-expert users and complex database querying processes. SQLucid addresses existing limitations by integrating visual correspondence, intermediate query results, and editable step-by-step SQL explanations in natural language to facilitate user understanding and engagement. This unique blend of features empowers users to understand and refine SQL queries easily and precisely. Two user studies and one quantitative experiment were conducted to validate SQLucid's effectiveness, showing significant improvement in task completion accuracy and user confidence compared to existing interfaces. Our code is available at https://github.com/magic-YuanTian/SQLucid.

UIST '24, October 13–16, 2024, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-0628-8/24/10
https://doi.org/10.1145/3654777.3676368

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Natural Language Interfaces, Databases, Explanations

ACM Reference Format:
Yuan Tian, Jonathan K. Kummerfeld, Toby Jia-Jun Li, and Tianyi Zhang. 2024. SQLucid: Grounding Natural Language Database Queries with Interactive Explanations. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24), October 13–16, 2024, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3654777.3676368

1 INTRODUCTION

The rise of big data has led to a growing demand for querying databases for data analysis and decision-making. To fully unleash the analytical power of databases, many natural language (NL) interfaces [18, 24, 54, 76] have been developed, enabling non-experts to express and fulfill their goals through NL queries. The backbone of these interfaces is a computational approach that translates an NL query to a database query in a formal language such as SQL. Early work in this domain applied rule-based or grammar-based approaches [34, 85]. Recent advances in deep learning have led to

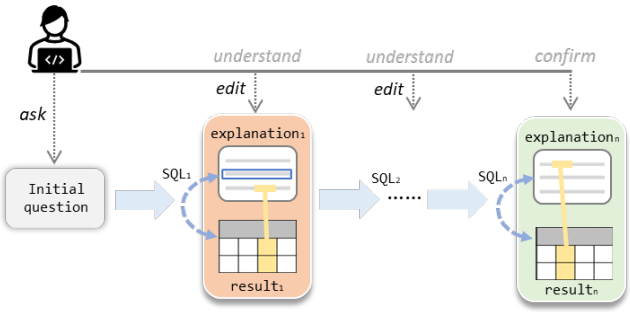

Figure 2: The iterative SQL refinement pipeline of SQLucid

a variety of text-to-SQL models [59, 72, 78, 86], achieving unprecedented performance on NL querying tasks.

Despite these great strides, text-to-SQL models cannot always reliably generate correct queries aligned with user intent. As a result, users run the risk of receiving wrong query results and henceforth making incorrect or suboptimal decisions. This is critical in high-stakes domains such as finance and healthcare. The leaderboard of a popular evaluation text-to-SQL benchmark, Spider[1], indicates that even with the best system [59] built on GPT-4 still suffers from an error rate of 10%. It is crucial to help users identify and fix the potential errors in the database queries generated by these models to avoid incorrect or suboptimal decisions. To bridge the gap, several approaches have been developed to enable users to provide feedback to SQL generation in an interactive manner [29, 42, 46, 54, 81].

However, most approaches only support feedback in constrained forms, e.g., answering multiple-choice questions [24, 46, 81], or changing keywords using a drop-down menu [54]. Such constrained feedback is insufficient to fix complex errors in real-world tasks. Ning et al. [55] conducted a user study to evaluate three representative approaches for SQL generation and refinement, including MISP [81], DIY [54], and SQLVis [52]. They found no statistically significant difference in user performance compared to manually writing SQL queries. Particularly, participants found it hard to understand the generated query and provide feedback.

To address this challenge, we draw inspiration from the grounding theory in communication [12]. The theory suggests that effective communication requires a common ground, where speakers design utterances for listeners to understand and listeners provide feedback to resolve ambiguity and demonstrate understanding. However, recent studies in code generation have highlighted challenges due to insufficient communication between systems and developers [5, 7, 70]. Systems often misinterpret the developer's intent, while developers often struggle to comprehend the generated code. This communication gap, which arises from a lack of common ground, results in code that does not align with user intent and hinders effective feedback [5].

Based on this insight, we develop an interactive system, SQLucid, that leverages step-by-step SQL explanations as the common ground between SQL generation models and users. Figure 2 provides an overview of the interaction pipeline. In each iteration, SQLucid generates an explanation in NL to describe the individual steps in the generated SQL query. Through the rich interaction mechanisms provided by SQLucid, users can quickly navigate the explanation to understand the query and verify its behavior. If users recognize any erroneous steps, they can directly edit the explanation to inform the model which part of the SQL query should be regenerated and what the expected behavior is.

Compared with existing techniques, SQLucid has two key features, visual correspondence and intermediate query results. First, without an efficient way to navigate the database, users could easily become overwhelmed by the volume of data and complexity in the schema. Visual correspondence helps users instantly locate the related data by interacting with the entities mentioned in the explanation. They can also mentally connect elements mentioned in the explanation with elements in the database, which is helpful for sense-making. Second, the complex database schema makes certain query operations difficult to intuitively explain in NL. This is due to a logic gap between human understanding and database operations [66]. For instance, explaining a `JOIN` operation to users based on primary and foreign keys can be challenging. Rendering intermediate results provides a convenient way for users to understand and verify the function of each step.

Finally, we conducted a comprehensive evaluation of the usability of SQLucid. This included two user studies with 38 participants in total and a quantitative experiment with 100 tasks. The first user study compared SQLucid with MISP [81] and DIY [54], demonstrating the effectiveness of our design choices over alternative designs. The second user study measured the contribution of each key feature in SQLucid, showing that each feature significantly improves usability. The quantitative experiment shows the generalizability to a broad range of querying tasks. The results indicate that accuracy improves from 49% when no interaction is possible, to 89% when using SQLucid and the user is familiar with it.

2 RELATED WORK

2.1 Interactive Support for Text-to-SQL

There is a large body of literature on converting natural language (NL) questions to SQL queries, ranging from logic-based [22, 74], rule-based [3, 42, 58, 62, 80] to neural-based methods [28, 60, 63, 72, 86]. However, these techniques only focus on improving the accuracy of text-to-SQL methods, instead of designing interactions to help non-experts understand and improve the query.

We summarize existing interactive support for text-to-SQL generation into two categories—(1) *explaining generated queries back to users* and (2) *soliciting user feedback to refine queries*. QueryVis [41] and SQLVis [52] explain SQL queries by visualizing them as graphs. However, graphical representations can become unintuitive and overly complex for end-users [55]. Many existing systems resort to explanations in NL instead [36, 37, 43, 54, 77]. For instance, Xu et al. [77] first convert an SQL query into a directed graph and then use a graph-to-sequence model to generate an NL summary of the query. DIY [54] uses pre-defined templates to translate an SQL query into a step-by-step explanation in NL. Similar to DIY [54], SQLucid also leverages step-by-step explanations in NL but uses a different grammar-based method. The benefit is that such a grammar-based method can handle arbitrarily complex queries without being restricted to pre-defined templates. Furthermore, existing systems

[1] https://yale-lily.github.io/spider

generally present explanations as static text, offering limited interactive capabilities for users to understand, validate, and refine queries. DIY [54] attempts to improve clarity by rendering intermediate results on a "small-but-relevant" sample database, but both their study [54] and the study by Ning et al. [55] indicate that this approach can lead to user confusion. Specifically, users may find inconsistency between query results on the sampled database with the full database, as some relevant data may be missing. To address this issue, SQLucid renders intermediate query results by executing on the entire database to make users fully comprehend the functionality of each step. Additionally, we propose further incorporating rich interactions in SQLucid such as visual correspondence and direct query editing to augment the utility of explanations, thereby enhancing user engagement and understanding.

A common way to solicit user feedback is through conversations [19, 24, 46, 81]. For instance, MISP [81], DialSQL [24], and PIIA [46] detect a set of tokens with high uncertainty during the decoding process and ask multiple-choice clarification questions to users. In these systems, users can only passively clarify their intent by selecting from a limited set of options in the multiple-choice questions. NL-EDIT [19] allows users to proactively suggest SQL query edits via free-form text. Then, it uses an encoder-decoder model to convert the free-form text to a sequence of edits to refine the query. Despite the flexibility, incorporating such open-ended feedback is challenging. It requires the model to precisely infer which parts of the query to edit and which edits to apply.

Direct manipulation [65] is an effective mechanism for rapid and accurate user feedback. Several text-to-SQL systems support direct manipulation and allow users to directly refine a query without knowing SQL syntax [20, 43, 54, 64, 68]. DIY [54], DataTone [20], and NaLIR [43] allow users to directly change table names, column names, and values used in a query via a drop-down menu. In Eviza [64] and Orko [68], users can adjust a numeric value in a query using a slider. However, these systems only support a limited set of simple edits to SQL. They do not allow users to specify complex feedback, e.g., selecting data from two tables (i.e., `JOIN`), grouping a set of data records (i.e., `GROUP BY`), etc.

Our idea of grounding database queries with explanations resembles a recent work by Liu et al. [49]. Liu et al. propose to use step-by-step explanations as a grounded abstraction to generate and refine Python code for spreadsheet data analysis. Despite the similarity in spirit, our work has several key differences in system design. First, like other text-to-SQL systems that explains SQL in NL, Liu et al. also only render the explanations as static, plain text. Compared with spreadsheet data analysis, database querying often involves complex data schema, multiple tables, and complex operations. Thus, our work presents richer interaction mechanisms to facilitate query comprehension and validation. Second, given user edits to a step-by-step explanation in NL, SQLucid performs fine-grained query refinement at the clause or entity level, without the need to regenerate the entire query from scratch. By contrast, their system concatenates the explanations of individual steps as a new prompt and invokes CodeX [10] to regenerate the entire Python code. This limits the utility of user feedback on step-by-step explanations and does not afford precise code refinement.

2.2 Human-AI Collaboration

Promoting efficient collaboration between intelligent systems and humans has been a long-standing research topic in HCI. This concept was first introduced in the seminal work on man-computer symbiosis [48]. In that work, Licklider proposed that computers could perform routine tasks to pave the way for human insights, while human users could utilize their domain knowledge to make decisions that computers are not capable of making. Nowadays, the inaccuracy of AI models in high-stake domains further necessitates collaboration between humans and AI. However, the lack of interpretability and communication convenience presents a significant challenge to effective human-AI collaboration [61]. Even though humans have the potential to complement AI, they often struggle to understand AI's states and effectively express their thoughts [4, 17, 35, 47, 49, 50]. Specifically, if a user does not understand where the error is and what causes the error, they may find it difficult to provide effective instructions on fixing the error [66].

Research from various domains has focused on explaining system behavior [14, 16, 21, 26, 51]. For example, Head et al.'s work on Tutorons [26] automatically generates context-relevant, on-demand in-situ explanations for code snippets, such as regular expressions, on web pages. While Tutorons aims to bridge the gap between programmers and complex syntax, SQLucid is designed for non-programmers to iteratively refine SQL queries in NL. Furthermore, SQLucid makes these explanations editable in free-form NL, enabling intuitive user feedback. SQLucid also incorporates visual correspondence and intermediate results to deepen user engagement compared to the static explanations provided by Tutorons.

3 USER NEEDS AND DESIGN RATIONALE

3.1 User Needs in SQL Generation

To understand the needs of non-experts when querying databases, we conducted a literature review of previous papers that have done a formative study of text-to-SQL systems [52, 54], have done a user study of existing tools [54, 55], or have discussed the challenges and opportunities of text-to-SQL systems [1, 8, 13, 30, 53, 66]. Based on this review, we summarize three major user needs.

N1: Users need effective methods to understand and validate a generated SQL query, so they can trust the result. Text-to-SQL systems are primarily designed for non-experts who are not familiar with SQL. Without additional support, the only way for them to validate the correctness of a generated query is to carefully examine if the query result looks reasonable. However, if a query involves too many rows, columns, and tables, it is cognitively-demanding and time-consuming to manually examine the query result. Kim et al. [66] point out that for queries that return a large amount of data, it is useful for users to understand how the resulting data is retrieved from the database. So users can reason about the correctness of the query steps, rather than a large amount of resulting data. Jagadish et al. [30] argue that database systems can frustrate users if there is no explanation for some unexpected query results. In a user study with 12 participants, Narechania et al. [54] found that participants appreciated explanations and wished to have multi-modal explanations to help them understand complex query operations, such as table joining and compound SQL clauses. Therefore, it is critical to help users validate the query behavior.

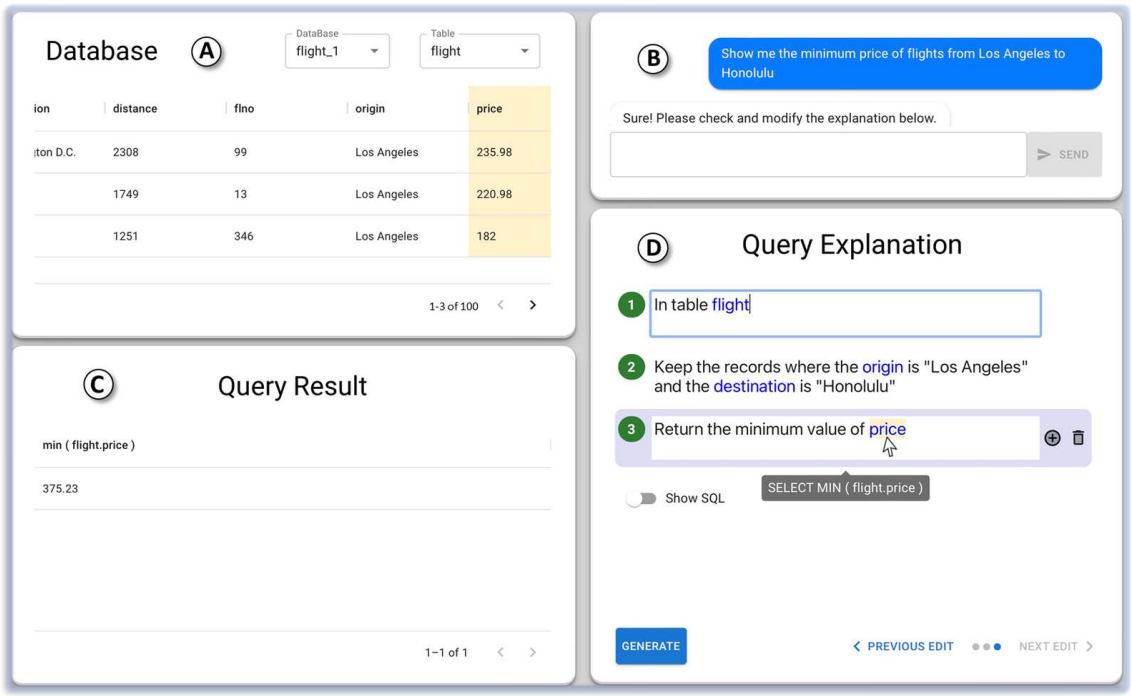

Figure 3: The user interface of SQLucid. (A) The *Database* panel allows users to switch databases and tables in a database. It also allows users to manually inspect, search, and filter data. (B) The *Question* panel allows users to ask a question to the database in natural language. (C) The *Query Result* panel shows the query result as well as the intermediate result of individual steps when the user clicks each step. (D) The *Query Explanation* panel renders the step-by-step SQL explanation in natural language. Users can directly edit the explanation to fix the incorrect behavior in a step, add new steps, or delete existing steps.

N2: Users prefer SQL explanations that are concise, well-organized, and intuitive. Ning et al. [55] compared SQL explanations generated by three interactive systems in a user study. They found that the majority of participants preferred the shorter explanations provided by DIY [54], since they are easier to read and understand. Leventidis et al. [41] argued that the explanation in NL can become very lengthy and verbose for complex SQL queries, limiting their readability and utility in practice. This is supported by a controlled lab study with 112 CS undergraduate students [8]. The study found that students can easily get lost when dealing with long and complex queries. When presenting a query in a more structured and succinct manner, students experienced significantly less cognitive load and performed much better in data query tasks. Therefore, we need to find the right level of abstraction that can concisely summarize the behavior of a SQL query in a clear and well-organized manner while matching user expertise.

N3: Users need more flexible and expressive ways to provide feedback. Most existing systems only support feedback in constrained forms, e.g., answering multiple-choice questions [24, 46, 81], changing incorrect keywords in a drop-down menu [54]. This hinders users' ability to handle various SQL errors, especially for those requiring a completely new clause or subquery. As shown by a recent study [55], such interactive mechanisms did not significantly improve the task completion rate or reduce the task completion time in complex text-to-SQL tasks compared with manually fixing a SQL query. Participants expressed frustration when they found they could not fix an error using the assigned feedback mechanism. Thus, non-expert users need a more expressive and flexible way to guide the model to fix various SQL generation errors.

3.2 Design Rationale

To support **N1**, we choose natural language as the communication vehicle, since it is understandable for non-experts and it is also flexible to express any kind of feedback. An alternative design is to explain a query in a graphical representation [41, 52]. While graphs can be visually appealing, they can also become overly complex and counter-intuitive for non-experts [55].

To support **N2**, we adopt step-by-step explanations and augment them with visual correspondence and intermediate query results. Users can utilize the visual correspondence to quickly locate relevant data and navigate a large database. This is particularly helpful when users are not familiar with the database schema or when there are many tables and columns. Displaying intermediate results helps users further validate the query behavior on concrete data and understand how each step contributes to the final result.

To support **N3**, SQLucid enables users to specify the correct behavior of a query step by directly editing the description of that step. There are several alternative designs for this feature. First, alternatively, we could ask users to rephrase the original NL question

(i.e., *prompt engineering*) or provide NL feedback in a conversation [19]. However, recent studies show that prompt engineering is challenging and accurately interpreting NL feedback is as hard as interpreting the initial NL query [18, 55, 84]. Some systems [20, 43, 54] also enable users to provide feedback via direct manipulation. As discussed in Section 2.1, these methods can only support limited edits. Another design option is to allow users to pinpoint errors in the intermediate or final query results. This design is effective for certain errors such as including extra columns and when the dataset is small. However, when the query results include many data records, it can be cumbersome and time-consuming to inspect all data and annotate which data records are wrong. Furthermore, regenerating the query based on input and output data may lead to overfitting, a known issue in programming-by-example techniques [39, 56].

4 SYSTEM IMPLEMENTATION

In this section, we first describe the base text-to-SQL generation model used in SQLucid and a pilot study to understand the usability issues of the base model. We then detail the SQL generation process in SQLucid and highlight three key features that facilitate efficient SQL query comprehension, validation, and repair.

4.1 Base Model and Pilot Study

The design of SQLucid is model-agnostic. SQLucid is built upon a SQL generation system called STEPS [69], which provides algorithms and technical components to generate SQL explanations but only provides limited interaction support. We choose STEPS since it generates reliable grammar-based SQL explanation and provides a pre-trained text-to-clause model for SQL regeneration. However, one can replace STEPS with other models, e.g., using GPT-4 to generate SQL and step-by-step explanations.

STEPS provides a limited primitive UI without careful consideration of the usability challenges. It only supports add or removing explanation steps. To better understand the usability challenges in STEPS, we conducted a pilot study with three participants. Each participant completed five data query tasks randomly selected from Spider [83], with an average task completion time of 4 minutes. We subsequently interviewed participants about their experiences.

The study revealed challenges in comprehending SQL through natural language descriptions alone. Participants found it time-consuming and cumbersome to manually navigate the database content, especially when explanations referenced multiple tables and columns. Participants often had to switch between tables and manually locate mentioned columns, making it difficult to validate query behavior. Participants also struggled to understand complex operations like JOINs and the concepts of primary and foreign keys, particularly in queries involving multiple tables. These findings highlighted the need for more intuitive ways to connect explanations with database entities and visualize query operations. It is important to ground the SQL explanations on the data to help users better comprehend and validate the entities and operations mentioned in the explanation. To address the usability issues, SQLucid proposes rendering visual correspondence, as detailed in Section 4.3, and displaying intermediate query results, as detailed in Section 4.4.

4.2 SQL Query and Explanation Generation

Given a SQL query generated by the underlying model, SQLucid generates a step-by-step explanation in NL for the query. Following STEPS, we use the same text-to-SQL generation model, SmBoP [60]. Yet users can plug in any model they prefer to use. Furthermore, we adopt the same grammar-based explanation generation algorithm of STEPS [69]. The SQL query is decomposed into SQL clauses and each clause is then translated into NL descriptions based on SQL grammar. This algorithm guarantees a deterministic and accurate translation from the SQL query to the NL description. Please refer to the STEPS paper [69] for technical details.

For users who know SQL, we still provide the option to view the generated SQL by clicking a toggle button below the explanation (Figure 6 ⓓ). This feature was requested by pilot study users who knew SQL and wished to double-check SQL code in our iterative design process. It is not designed for non-experts, since they are not familiar with SQL syntax and semantics. SQLucid. Reading NL descriptions and checking intermediate results is the main way for non-experts to validate SQL.

4.3 Visual Correspondence via Highlighting

As illustrated in Figure 4, SQLucid highlights the noun phrase of each database entity in blue in the SQL explanation. We chose blue as it is the standard color for hyperlinks, which implies the highlighted entity can be interacted with. When users hover over a highlighted entity, SQLucid will automatically navigate to the corresponding data in the database panel and highlight the corresponding data. Specifically, if the entity is a table, the drop-down menu turns green to indicate that the table is in focus (Figure 4 Ⓐ). If the entity is a column, SQLucid automatically centers and highlights the column in yellow (Figure 4 Ⓑ).

A special case is nested SQL queries. The explanation generator of STEPS [69] splits a nested query into subqueries and generates an explanation for each of them separately. When Subquery A uses the result of Subquery B, the explanation of Subquery A will refer to the query result of Subquery B in natural language. To help users easily recognize which subquery's result is used by another subquery, SQLucid highlights the NL references with underscored hyperlinks. Section 5 illustrates this scenario.

We leverage the explanation generation method to establish the initial entity mappings. In particular, we instrument the explanation generator to log the translations between database entities and noun phrases in the explanation. SQLucid buffers these mappings in memory and dynamically highlights the database content when users hover over a noun phrase that maps to a database entity. Additionally, for nested SQL queries where one query may refer to the result of another, SQLucid uses natural language description (e.g., "result of the first query") to reference the result generated by a previous query. SQLucid also establishes a mapping and visually shows the correspondence between the reference and the previous result. Since SQLucid allows for free editing of the explanation in NL, users may rephrase some names, introduce new names, or even make typos. Whenever the explanation is edited, SQLucid re-calculates the mappings based on text similarity.

We explored various approaches to calculate text similarity between database entities and SQL explanations. Initially, we considered using cosine similarity with word embeddings to better

Figure 4: (A) Hover over a *table name* and SQLucid automatically switches to the corresponding table and highlights it. (B) Hover over a *column name* and SQLucid automatically highlights the entire column.

capture semantic relationships. However, we found that leveraging semantic information was often inaccurate when dealing with abbreviated or similar database entity names. Consequently, we opted for Levenshtein distance due to its optimal balance of precision and computational efficiency. Levenshtein distance can effectively capture nuances in spelling variations and similar names, while its lightweight nature ensures quick responsiveness in SQLucid.

4.4 Intermediate Query Results

In order to help users understand the purpose of each step, SQLucid allows users to view the intermediate results corresponding to each step. To compute the intermediate result, SQLucid synthesizes a temporary SQL query by combining the current step with all preceding steps. However, simply concatenating SQL clauses from these steps may result in syntax errors or incomplete queries. For instance, missing the `SELECT` clause leads to an invalid SQL. To address this issue, we developed a synthesis algorithm inspired by the grammar-based explanation generation algorithm in STEPS [69]. This algorithm converts a sequence of explanation steps back into a SQL query while following the grammar rules. Any missing clauses are automatically populated with dummy placeholders (e.g., `SELECT *`). The resulting temporary SQL query is then executed on the database to compute the intermediate result.

When the user clicks on the circled number of each step, the background of the corresponding step will turn blue, indicating this step is selected. The *Query Result* view (Figure 3 Ⓒ) is then updated to show the intermediate query result.

Figure 5 demonstrates an example.When the user selects the first step, the database returns all the records in table *flight*, with an initial temporary SQL query of "`SELECT * FROM flight`". When the user selects the second step, the database filters out all the records in the first step that do not satisfy this condition (i.e., flight from Los Angeles to Honolulu). The temporary SQL query becomes "`SELECT * FROM flight WHERE flight.origin = "Los Angeles" AND flight.destination = "Honolulu"`". When the user selects the third step, the database returns the minimal price from the remaining records. The temporary SQL query becomes "`SELECT MIN (flight.price) FROM flight WHERE flight.origin = "Los Angeles" AND flight.destination = "Honolulu"`".

4.5 Query Refinement by Explanation Editing

While inspecting the explanation in NL and the intermediate query results, if a user finds an erroneous step, they can directly edit the description of that step to specify the correct behavior (Figure 6 ⓒ). Users can type in any description in free-form text, without being confined to a certain format. Users can also add a new step at any position or delete any existing step by clicking on the "Add" or "Delete" button next to an existing step (Figure 6 ⓑ). Once the user has finished modifying the explanation, they can click the "Generate" button to request SQLucid to regenerate the SQL based on the edited explanation (Figure 6 ⓔ). A complex SQL query can sometimes consist of multiple subqueries (a SQL statement with only 1 SELECT keyword) concatenated together with set operations (e.g. UNION). Within a single subquery, the position of a newly added step is not important in our design, as SQLucid can reorder and rectify all steps based on clause types to form a valid subquery. However, for a complex query involving multiple subqueries, users should ensure that new steps are added to the explanation of the corresponding subquery. Finally, SQLucid allows users to check their edit history and undo/redo some edits by clicking on the stepper buttons at the bottom (Figure 6 ⓕ).

To interpret the edited explanation and correct the error, we adopt the same text-to-clause model used in STEPS [69]. It achieves an exact match accuracy of 90.6%. Like the text-to-SQL model, this model is independent of our system and can easily be replaced by other models. After regenerating a clause, SQLucid merges it with the original query and automatically rectifies any syntax errors or conflicts. Please refer to the STEPS paper for more technical details.

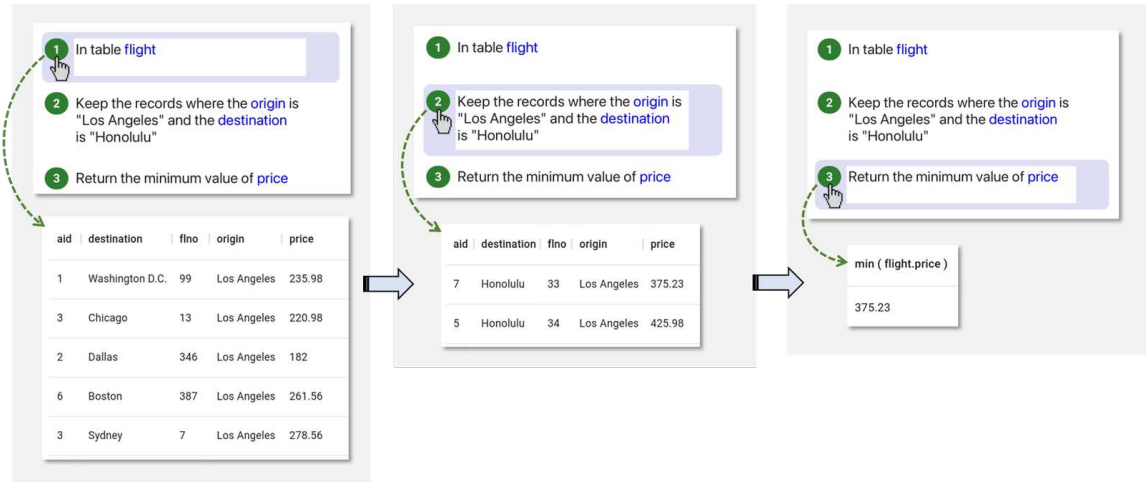

Figure 5: When clicking on the step number, the *Query Result* view will visualize the intermediate result after the selected step.

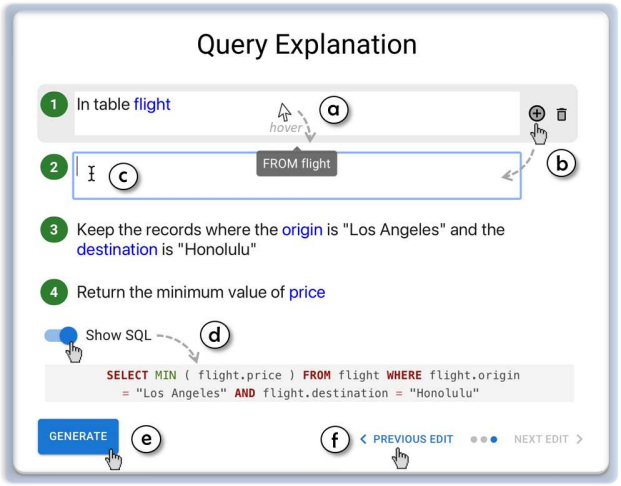

Figure 6: Interacting with the step-by-step explanation

5 USAGE SCENARIO

Suppose Alice is a social scientist and she wants to investigate the correlation between people's mobility behavior patterns and flight prices since the pandemic. She needs to analyze a large database with millions of flight records distributed in many different tables. As the first step, she wants to know the airport with the most flights to the most popular destination in the first quarter of 2022. Alice finds it time-consuming to manually filter the database and find the desired data record. Furthermore, since the information spans across multiple tables, Alice does not know how to filter the data based on multiple conditions on multiple tables simultaneously.

Therefore, Alice decides to try SQLucid. She asks, "*Show me the airport which has the most flights to the most popular destination in the first quarter of 2022.*" in the *Question* panel. Then, based on Alice's question, SQLucid automatically generates a SQL query and executes it in the database. Alice looks at the query result but she is not sure whether it is correct. Therefore, Alice chooses to read the step-by-step explanation of the SQL query in the *Query Explanation* panel. Since the generated query is a nested query, SQLucid explains the inner query as the first and the outer query as the second query below:

Start the first query

(1) Merge data in table flight and table travel.
(2) Keep the records where month is January.
(3) Split the data into groups based on the destination.
(4) Sort the groups based on the number of records in descending order, and return the first record.
(5) Return the destination.

Start the second query

(1) In table travel.
(2) Keep the records where the destination is the result of the first query.
(3) Split the data into groups based on the airport code.
(4) Sort the groups based on the number of records in descending order, and return the first record.
(5) Return the airport name.

The step-by-step SQL explanation gives Alice a high-level understanding of the generated SQL query. She roughly understands the purpose of the first query is to find the most popular destination, and the purpose of the second query is to find the airport with the most flights to this destination (hyperlinked blue text in Step 2).

However, Alice is unsure about what kind of code is associated with "*airport code*" in Step 3 of the second query. Thus, she wants to see some actual data in the database. However, when she tries to locate the related data in the database, she notices there are many tables and some tables even include hundreds of columns. She does not want to do this manually. Instead, Alice hovers her mouse over the highlighted text "*airport code*" in this step. The database panel automatically switches to the table that includes "*airport code*", centering and highlighting data in this column in yellow. After reviewing the data in the database, Alice confirms

that the airport code is a unique identifier used when booking a flight. She is confident this step has no issue.

Alice is curious about the difference between Table "*flight*" and Table "*travel*" in Step 1 of the first query, so she hovers the mouse over these two entities respectively. By moving the mouse between them, the corresponding tables are being switched accordingly. Alice clearly notices why these two tables need to be merged. This is because "month" is stored in the Table "flight", while "destination" is stored in Table "travel".

Since Alice is not familiar with SQL, she still does not understand how these two tables are merged in this step. Therefore, she clicks on this step to view the intermediate result. The intermediate result shows a combined table with columns from the "*flight*" table and the "*travel*" table. Alice checked a few data records in the merged table and compared them with the original records in these two tables to confirm that they were indeed consistent.

As Alice reads individual steps in the SQL explanation, she notices that Step 2 of the first query is wrong. It seems SQLucid misinterpreted the meaning of "the first quarter" as "January", and it also ignored the year constraint. Instead of rephrasing her original question, Alice modifies the description of Step 2 in the first query by explicitly specifying the beginning and ending months. She then adds a new step below to instruct the system to only consider data in Year 2022. Below is the modified SQL explanation.

Start the first query

(1) Merge data in table flight and table travel.
(2) Keep the records where month is [January] → [**between January and March**]. (Updated)
\+ (3) Make sure the year in 2022. (Added)
(4) Split the data into groups based on the destination.
(5) Sort the groups based on the number of records in descending order, and return the first record.
(6) Return the destination.

Start the second query

(1) In table travel.
(2) Keep the records where the destination is the result of the first query.
(3) Split the data into groups based on the airport code.
(4) Sort the groups based on the number of records in descending order, and return the first record.
(5) Return the airport name.

Then Alice clicks on the *Generate* button to update the query. She receives a new airport name and a new SQL explanation. By checking the explanation and the intermediate results again, Alice is convinced that the result is correct and exactly what she needs.

6 USER STUDY I: COMPARISON WITH OTHER INTERACTIVE APPROACHES

To investigate the usability of the holistic system, we conducted a within-subjects user study with 30 participants in comparison to two representative interactive systems, MISP [81] and DIY [54]. To ensure a fair comparison, we have redesigned the front-end user interfaces of MISP and DIY following the same design style as SQLucid (detailed in Appendix C). Furthermore, we have replaced the original SQL generation model in MISP and DIY with the same SQL generation model used in SQLucid. In this way, we normalize the impact of the visual appearance and also the underlying models on user performance in the comparison.

6.1 Participants

We recruited participants through the mailing lists in an R1 university. To investigate the impact of user expertise on SQLucid, we selected participants with three different levels of familiarity with SQL. In total, we recruited 30 participants. 15 of them had never heard about or used SQL before (*end-user*); 10 knew the basics of SQL but had to search online to recall syntax details (*novice*); 5 could fluently write SQL queries (*expert*). 14 participants were undergraduate students, 4 were master's students, and 12 were PhD students. We shared the consent form in the recruitment email and obtained their consent before each study. Each participant received a $25 gift card as compensation for their time.

6.2 Comparison Baselines

MISP [81] and DIY [54] are two state-of-the-art interactive approaches for SQL generation. They adopt two typical mechanisms, question-answering and direct manipulation.

MISP uses a question-answering interaction mechanism, where users clarify ambiguities through multiple-choice questions. To enable fair comparison, we created a graphical interface for MISP similar to SQLucid, excluding the *Query Explanation* view (Figure 3 Ⓓ), and used the same text-to-SQL model [60] as SQLucid.

DIY employs direct manipulation, allowing users to correct mappings between SQL tokens and natural language phrases using drop-down menus. We adapted the replication from Ning et al. [55] with a SQLucid-like interface and the same underlying model [60].

Appendix C provides details and screenshots of baseline UIs.

6.3 Tasks

We performed stratified random sampling on a widely used text-to-SQL benchmark, Spider [83], to create a pool of 48 tasks. This task pool includes 12 easy tasks, 12 medium tasks, 12 hard tasks, and 12 extra hard tasks, according to the difficulty classification from Spider. Table 6 in the Appendix show 12 representative tasks.

6.4 Protocol

Each study consisted of three sessions, one for each tool. We randomized the order of assigned tools to mitigate learning effects. Each session starts with participants watching a tutorial video about the assigned tool. Then participants were given several minutes to practice and get familiar with the tool before working on real tasks. Once they were done practicing, participants were given 10 minutes to complete 8 assigned SQL tasks using the designated tool. Specifically, we selected 2 tasks per difficulty level from the pool of 48 tasks. We randomized the order of the 8 tasks in each session to counterbalance the impact of task difficulty levels (e.g., doing easy tasks first vs. doing difficult tasks first). If a participant found a task too difficult to solve, they were allowed to skip it. For each task, participants were asked to first read the task description and then ask an initial natural language question to the assigned tool. After receiving the generated query and the query result, the participant

could further validate and correct the generated query using the interaction mechanisms provided by the tool.

At the end of each session, participants were asked to complete a post-task survey to share their experiences. The survey included the NASA Task Load Index (TLX) questions [25] and several 7-point Likert-scale questions to rate their perception of the assigned tool. After all sessions, participants completed a final survey, in which they directly compared all the tools and shared their overall thoughts about the usefulness of the tools. We recorded each study with the permission of the participants. Participation took 79 minutes overall on average.

6.5 User Performance

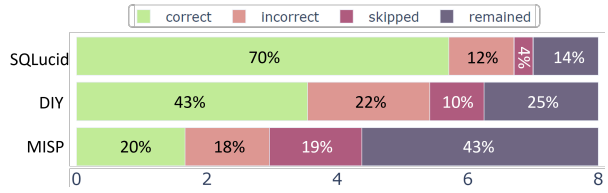

Figure 7: Distribution of correctly completed, incorrectly completed, skipped, remaining tasks (Study 1)

Table 1: Task Completion Accuracy (Study 1).

	Task completion accuracy	SD
MISP [81]	56%	30%
DIY [54]	67%	20%
SQLucid	**85%**	13%

Task Completion Rate. Figure 7 shows the distribution of completed, correct, skipped, and remaining (i.e., tasks that were not even tried due to the time limit) tasks when using different tools. An ANOVA test showed that the mean differences among the number of completed tasks, correct completion, skipped tasks, and remaining tasks when using different tools are all statistically significant (p-value = 1.75e-16, 7.99e-25, 7.62e-6, 1.21e-2 respectively).

Specifically, participants using SQLucid completed 6.6 out of 8 tasks, while participants using MISP and DIY completed 3.0 and 5.4 tasks respectively. This result suggests that SQLucid can accelerate the speed of task completion. Furthermore, when using SQLucid, participants skipped only 4% of the tasks, compared with 10% when using DIY and 19% when using MISP. This implies that SQLucid can provide more effective support to help participants make progress on challenging tasks, leading to fewer skipped tasks.

To measure the correctness of completed tasks, we calculate the *task completion accuracy*—the number of correctly completed tasks divided by all completed tasks, excluding skipped tasks and remaining tasks. Table 1 shows the result. Participants using SQLucid also achieved the highest task completion accuracy, 85%. In contrast, participants using MISP and DIY only achieved 56% and 67% accuracy, respectively. In other words, in 44% and 33% completed tasks, participants using MISP and DIY thought they had arrived at a correct query when in fact, the query was still wrong. These results imply that SQLucid can significantly improve user productivity when querying databases and help them effectively recognize query errors and generate correct queries with high accuracy.

Utility Rates of Different Features. To better understand the utility of different features, we analyzed recordings and gathered utility rates of features. For each task, participants intentionally navigated data by checking the visual correspondence 10.2 times. Participants rendered the intermediate results 3.5 times. In 48% of assigned tasks, SQLucid generated the correct query in the first iteration and participants did not edit the SQL explanation. In 47% of assigned tasks, SQLucid generated a wrong query in the first iteration, and it took 1.8 edits to fix. In 5% of assigned tasks, participants either rephrased the question or skipped the task.

These values show that participants heavily rely on visual correspondence and the intermediate results to understand the query. With these features, participants can quickly identify and successfully fix errors with only a few edits per task, improving the initial query generation accuracy from 48% to 85%. We also analyzed the recordings of participants using DIY and MISP. We found that DIY and MISP generated the initial query correctly in 48% and 51% of the assigned tasks. However, due to the limitation of their interaction methods, participants could not effectively understand the generated query and only fixed a limited number of queries, resulting in a 56% and 67% final accuracy, respectively.

The Impact of User Expertise. Table 2 shows the number of correctly completed tasks for participants with different levels of expertise. Overall, compared with MISP and DIY, SQLucid consistently improved the task completion correctness and efficiency across all levels of SQL expertise. Specifically, the performance gap between different expertise levels when using SQLucid is narrow. An ANOVA test showed that when using SQLucid, there is no statistically significant difference in the number of correctly completed tasks between different levels of SQL expertise (p-value=0.88). This implies that SQLucid can help bridge the expertise gap among users when querying databases.

Table 2: Correctly completed tasks by expertise level

	MISP [81]		DIY [54]		SQLucid	
	#Corr.	SD	#Corr.	SD	#Corr.	SD
End-User	1.6	0.91	3.3	1.13	5.4	0.91
Novice	1.4	0.67	3.5	1.27	5.7	1.06
Expert	2.2	1.10	3.8	1.30	5.9	1.22

The Impact of Task Difficulty Levels. Table 3 shows the number of correctly completed tasks at different levels of difficulty when using different tools. Overall, compared with MISP and DIY, SQLucid consistently improved the task completion correctness and efficiency across all levels of task difficulty. In particular, SQLucid significantly improves user performance on hard and extra-hard tasks. Compared with using MISP, participants using SQLucid completed almost 9X and 3X more extra-hard tasks correctly compared with using MISP and DIY. P10 wrote, "*I really enjoyed [SQLucid] a lot better than the previous two. I can use it to answer complex*

questions. Sometimes the system made a mistake at the first step, but I can easily correct it or add more constraints."

Table 3: Correctly completed tasks by difficulty level

	MISP [81]		DIY [54]		SQLucid	
	#Corr.	SD	#Corr.	SD	#Corr.	SD
Easy	0.81	0.74	1.40	0.69	1.62	0.50
Medium	0.49	0.62	1.19	0.72	1.48	0.60
Hard	0.21	0.51	0.54	0.61	1.39	0.58
Extra hard	0.12	0.31	0.35	0.63	1.14	0.66

6.6 User Confidence and Cognitive Load

In the post-task survey, participants self-reported their confidence about generated queries when using different tools on a 7-point scale. Figure 8 shows the distribution of users' confidence levels. The average confidence level is 6.42 when using SQLucid, compared with 3.79 and 5.29 when using MISP and DIY. An ANOVA test showed that the mean differences are statistically significant (p-value = 1.53e-11). Based on a qualitative analysis of user responses, we believe this improvement was largely attributed to the visual correspondence and intermediate features provided by SQLucid. P23 wrote, "*I felt most confident using SQLucid because it provided the most information on how a natural language query was interpreted and carried out. For example, I could see intermediate results and explanations of steps in natural language, allowing me to easily gauge whether the process was correct or not.*" P9 reported, "*When I was trying to explore the data for the other two tools, it was a bit challenging. But with related tables and data highlighted w.r.t. the explanations made it easier to navigate the data.*"

Figure 9 shows participants' ratings on the five cognitive load factors from the NASA TLX questionnaire [25]. The ANOVA test demonstrates that the mean differences are all statistically significant (p-value=8.26e-4, 7.83e-06, 6.04e-13, 2.57e-06, 8.10e-08 respectively). The result confirms that SQLucid can reduce users' cognitive load by creating interactive SQL explanations with visual correspondence and intermediate results, which serves as a common ground between users and the database. P19 wrote a comprehensive comment to illustrate the convenience provided by SQLucid—"*SQLucid helps me query the database and debug my query completely with natural language, which is good because I do not know SQL. The intermediate results help me locate bugs easily, so I don't need to debug my entire query. The natural language interpreter is so flexible that I do not need to change my writing style to accommodate it. All inferences are performed on the database level, so I don't need to specify which table I should look into. The highlight feature also helps me navigate the database.*"

6.7 User Ratings of Individual Features

In the post-task survey, we prepared six 7-point Likert scale questions for participants to rate the usefulness of key features in SQLucid. The most appreciated features were *being able to understand the SQL query via the step-by-step* explanation and *being able to directly edit the explanation in natural language to fix an error.* Other features in SQLucid ere also appreciated by the majority of participants. More discussion is detailed in Appendix A.

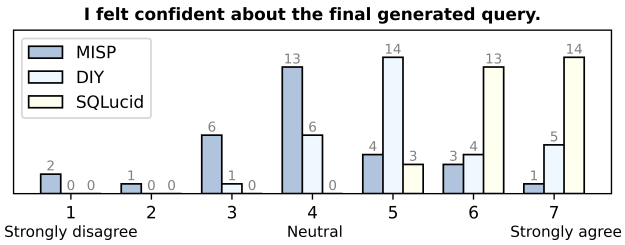

Figure 8: User Confidence Ratings (Study 1)

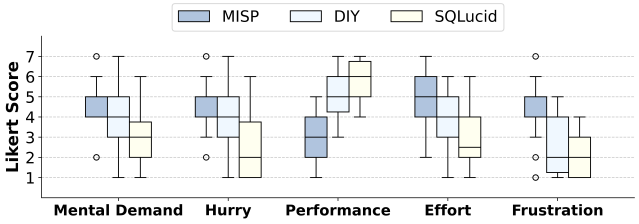

Figure 9: NASA Task Load Index Ratings (Study 1)

6.8 User Preference and Feedback

When asked about the tool they preferred to use for their real-world data query needs, all 30 participants selected SQLucid. We coded participants' responses in the post-study survey and identified two main reasons why they liked SQLucid more. First, 27 participants mentioned that the explanations provided by SQLucid were more understandable and useful. Particularly, the visual correspondence and intermediate result features bring more interactivity in SQLucid, and greatly enhance users' ability to identify errors.

Second, 21 participants pointed out that SQLucid is the most useful among all conditions because the direct editing of SQL explanations in NL is more convenient and requires less effort. P23 wrote, "*SQLucid was the most usable because I felt that it was very easy and fast to correct mistakes in interpretation using this tool. For example, I could directly use language to edit some of the intermediate steps to get the correct order of steps. I think this is fast and convenient.*"

In the post-task survey, we also asked participants what additional features may help them better solve the task. Seven participants mentioned that it would be helpful to see confidence scores associated with each step, because they can pay more attention to those steps with lower confidence. P1 wrote, "*I wish to see a confidence score that indicates if I need to check or debug something.*" Furthermore, three participants mentioned they would like to see some suggestions when editing the SQL explanation. P11 suggested that "*providing suggested expressions may diminish the chances for the normal language question to be misinterpreted.*" Finally, two participants mentioned that it might be useful for SQLucid to generate multiple answers and let the user choose one.

7 USER STUDY II: ABLATION STUDY OF KEY FEATURES IN SQLUCID

To investigate the effectiveness of each feature in SQLucid, we conducted another within-subjects user study with 8 participants, comparing SQLucid with three of its variants.

7.1 Participants, Baselines, Tasks and Protocol

We followed the same procedure as Study 1 to recruit 8 participants for this study. 4 of them had never heard about or used SQL before (*end-user*); 2 knew the basics of SQL but had to search online to recall details of the syntax when writing a SQL query (*novice*); 2 could fluently write SQL queries (*expert*).

We created three different variants of SQLucid as comparison baselines by ablating the two key features: (1) no visual correspondence, (2) no intermediate results, (3) no visual correspondence & no intermediate results (i.e., Text SQL explanation only).

In this study, we used the same tasks (Section 6.3) and followed the same protocol (Section 6.4) as the first user study. On average, each study took about 61 minutes in total.

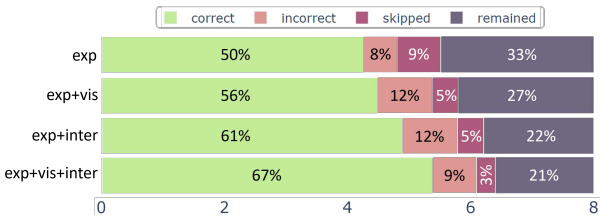

Figure 10: Distribution of correctly completed, incorrectly completed, skipped, remaining tasks when using different versions of SQLucid (Study 2)

7.2 User Performance

Figure 10 shows the distribution of completed tasks, correct tasks, skipped tasks, and remaining tasks. Table 4 shows the task completion accuracy similar to user study 1. An ANOVA test showed that the mean differences among these values are statistically significant, except for skipped tasks (p-value = 2.12e-02, 3.36e-02, 3.3e-01, 1.03e-03, 1.21e-2 respectively).

Specifically, when the SQL explanation is plain text, participants completed 4.9 out of 8 tasks with a completion accuracy of 81.6%, and skipped 0.75 out of 8 tasks. When the *visual correspondence* feature is activated, participants completed 5.5 tasks with a completion accuracy of 83.1% and skipped 0.375 tasks. When the *intermediate query result* feature is activated, participants completed 5.9 tasks with a completion accuracy of 83.5% and skipped 0.375 tasks. When both features were activated, participants completed 6.4 tasks with a completion accuracy of 84.3% and skipped 0.25 tasks. The result implies both the two features can reduce task completion time and increase user performance.

Table 4: Task Completion Accuracy (Study 2).

	Task completion accuracy	SD
Text Explanation Only	81.6%	7.9%
+Visual	83.1%	16.3%
+Intermediate	83.5%	11.9%
+Visual+Intermediate	**84.3%**	8%

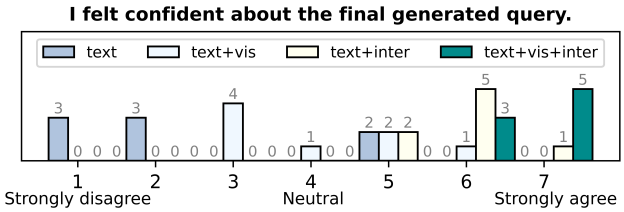

Figure 11: User Confidence Ratings (Study 2)

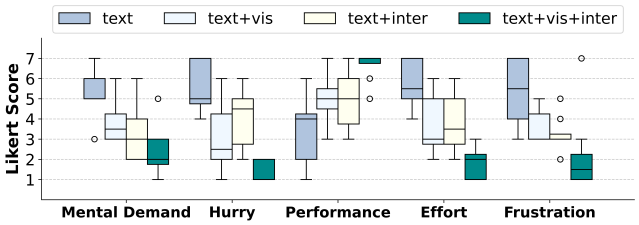

Figure 12: NASA Task Load Index Ratings (Study 2)

7.3 User Confidence and Cognitive Load

Figure 11 shows the participants' confidence with different tools. An ANOVA test shows that the mean differences across different conditions are statistically significant (p-value=1.32e-12). Figure 12 shows user ratings on the five cognitive load factors from the NASA TLX questionnaire. An ANOVA test shows that the mean differences in all five dimensions are statistically significant (p-value=1.65e-05, 2.71e-06, 1.14e-16, 7.34e-07, 2.53e-09 respectively). Participants using SQLucid with all features activated have the lowest cognitive load and highest confidence. The result shows both the *visual correspondence* feature and the *intermediate query result* feature serve as great supplements to the plain SQL explanations.

We analyzed the post-study survey responses and found that these two features contributed to different aspects of user performance. Specifically, the *visual correspondence* feature aids in data navigation, thereby saving more time. P1 wrote, "*[Without visual correspondence,] I need to use the scrolling bar a lot. That is annoying and tedious.*" On the other hand, the *intermediate query result* feature focuses on improving user comprehension of the explanation, which brings more confidence. P4 wrote, "*Intermediate results give me confidence about the final outcome.*" Additionally, this feature provides information that users may not have asked for, but can offer additional context, thereby reducing their cognitive load. P1 commented, "*Intermediate steps can help me check back and forth based on my needs. Without this feature, I only get a piece of information. If I want to know more, I need to ask multiple times.*"

Overall, features in SQLucid complement each other and collaboratively enhance the interactivity of SQL explanations. P2 made a comprehensive comment about the variant with only plain textual explanation, "*Without these features, my interest in using this system decreases a lot, because I need to find the data by my eyes and the mouse. Although it explains the procedure in English and provides the final result, I can't see the relationship between the sentences and the real data. Without seeing the relationship, it might be correct, but*

I question my understanding and do not trust it. Besides, sometimes when my request is too complex for the system to handle, I don't know which step is wrong."

8 QUANTITATIVE EVALUATION

To evaluate the generalizability of SQLucid , we further conducted a quantitative experiment where the first author completed 100 database query tasks. The results of this study can be interpreted as the upper bound of user performance of SQLucid.

We followed the same sampling strategy in the user studies, including 25 easy tasks, 25 medium tasks, 25 hard tasks, and 25 extra hard tasks from Spider [83]. For each task, the first author examined the database, read the natural language description, and tried to solve the task using SQLucid. The task was considered completed when a correct query result was obtained. This simulates an ideal condition where a user is familiar with the tool and has sufficient knowledge of database queries. This case study is to investigate to what extent SQLucid can solve query tasks, regardless of its learnability or usability.

The experimenter completed all 100 tasks with an accuracy of 89%, in an average of 1.9 minutes (median=0.9, SD=0.6) for each task. Table 5 shows the task completion accuracy at different levels of task difficulty. 5 out of 100 tasks were failed due to user misunderstanding. For example, while the correct SQL for *"French citizens"* should be "`Citizenship = France`" according to the data in the database, the query produced by the experimenter had "`Citizenship = French`". It is possible that additional affordances that proactively provide information about matches with content in the database could address these issues. 4 out of 100 tasks failed to be completed due to the complex query structure, e.g., a query with multiple subqueries. The experimenter decided to skip them because they were time-consuming to solve.

Table 5: Task completion at different levels of task difficulty

	Easy	Medium	Hard	Extra hard	Overall
Accuracy	96%	96%	76%	88%	89%

9 DISCUSSION

9.1 Design Implications

Based on the evaluation results, we found that the primary enabler for SQLucid lies in the bi-directional, natural language (NL) communication channel it establishes between human users and SQL generation models. Compared to directly editing and refining the original question (i.e., prompt engineering), editing the step-by-step explanations provides a more structured way to give feedback and allows users to pinpoint the error. Furthermore, by breaking down a lengthy explanation into shorter descriptions of individual steps, SQLucid can clearly and systematically explain the behavior of a query. The editability of these explanations allows human users to identify the specific step where an error occurs and directly propose a correction by altering the NL description of the erroneous step. This design enables users to offer more precise feedback and incrementally build a complex query than they could by providing high-level suggestions in a multi-turn dialogue (e.g., MISP [81], ChatGPT), thereby streamlining the SQL regeneration process.

The success of SQLucid also echoes the *grounding* theory in communication [12]. Grounding theory states that conversation is a collaborative effort aimed at establishing common ground or shared knowledge. In interactions with intelligent systems, such as SQL generation models, the system should offer evidence of understanding in response to a user's input, enabling the user to assess progress toward their goal. In our work, the editable step-by-step explanation serves as the common ground for communication between an SQL generation model and a human user—*the model explains a generated query step by step, while the human user corrects the model's misinterpretation by directly editing the explanation.* Furthermore, both the visual correspondence and the intermediate query result features further enhance the grounding.

Our work further illustrates that comprehending system behavior and repairing system breakdowns are highly interdependent activities. This is in line with previous studies of conversational agents [2, 6], which argue that users must first understand the current state of the system and the cause of a breakdown to choose an effective repair strategy. By providing a detailed explanation with intermediate results, SQLucid enables users to rapidly grasp the query's behavior and identify the root cause of an incorrect query result. This helps users to efficiently pinpoint the erroneous part of the query and give accurate and effective suggestions to fix it. Additionally, this design offers users greater flexibility in expressing their intent and feedback compared to relying on constrained mechanisms to gather feedback [29, 42, 46, 54, 81].

9.2 Using Interactive Explanation for Task Decomposition

Task decomposition is a long-standing challenge in program synthesis and code generation [23, 31, 39, 71]. Several approaches support task decomposition by asking users to specify intermediate steps [27, 33, 82]. For instance, Wranger [33] recommends a ranked list of operators at each synthesis step and asks users to select which operator to use and fill in the parameters. Using such systems requires users to be familiar with the underlying programming language and also actively think about intermediate steps to arrive at the final solution. Prior work shows that non-experts often find it difficult to decompose a complex task into sub-tasks [39].

The editable step-by-step explanation can serve as a scaffold to guide non-experts to decompose a complex task. Compared with prior work, SQLucid does not require users to actively make a task decomposition plan. Instead, the step-by-step explanation can be viewed as an initial decomposition plan proposed by SQLucid. Users only need to read and correct it. In particular, the step-by-step structure of the explanation will spontaneously inspire users to think about the intermediate steps and make it easier to recognize incorrect or missing steps. Since the explanation is communicated in natural language, users also do not need to know the semantics of the underlying programming language.

As we were developing this system, the rise of Large Language Models (LLMs) has brought another possibility for task decomposition. Recent studies have shown that LLMs are capable of

breaking down a large task into smaller subtasks with proper instructions [32, 57, 67, 73, 75]. For instance, Chain-of-Thought (CoT) Prompting [75] allows users to provide several examples of how to solve a problem analytically step by step and leverages the in-context learning capability of LLMs to decompose similar problems. Given a natural language query, one can use CoT to decompose it and generate a step-by-step plan with basic query operations. However, one caveat is that LLMs may hallucinate and generate an incoherent plan with non-sensical steps, as shown by many studies [38, 79]. In contrast, our grammar-based explanation method is strictly grounded in the SQL components and provides a faithful representation of computation steps in a query. We also provide a dedicated method to incorporate user refinement on individual steps to fix query generation errors.

9.3 Application to Other Domains

We believe that our interface design can be generalized to adjacent domains, such as enabling user validation and repair in code generation [11, 27], data transformation synthesis [15, 51], web automation [9, 40], smartphone app automation [44, 45], and regular expression synthesis [87]. Programs in these domains can be naturally decomposed into smaller components (e.g., program statements, API calls) and then explained in natural language in a similar step-by-step fashion. However, for certain domains such as tensor transformation synthesis [88], step-by-step explanations may not be the most suitable approach, as code in these areas often involves complex concepts and computation steps, such as linear algebra, which are challenging to clearly explain in natural language.

9.4 SQL Experts vs. Non-Expert Users

SQLucid is specifically designed for non-experts who need to interact with databases but lack SQL expertise. Reading NL descriptions and checking intermediate results is the main way for non-experts to validate SQL queries. Our analysis of user performance across varying SQL expertise levels reveals that the performance gap between end-users, novices, and experts has been substantially reduced when utilizing SQLucid. Our user study results show adding and removing NL steps are intuitive for non-experts. Users can freely edit the NL description of a query step and SQLucid updating the corresponding SQL component accordingly based on a text-to-clause model. If one step (e.g., group students into clusters by years) is missing in a query (e.g., compute the average GPA of students for each year), it is easy to recognize it from the NL description and the results.

While our focus was on non-experts, we discovered that SQLucid can also enhance the productivity of SQL experts. For complex tasks that necessitate joining multiple tables or creating compound queries, SQLucid offers a solid starting point from which SQL experts can iteratively and incrementally refine the query. For example, users can build two simple subqueries respectively and reference one within the other to form a more complex query.

Another unintended benefit was that participants in our study found SQLucid to be valuable for learning SQL. Five participants who were unfamiliar with SQL actively reported that their ability to read basic SQL queries improved as a result of using SQLucid, and they expressed a desire to continue using it for practical SQL learning. Participant P12 commented, "*It was nice to see the generated SQL code with human language. I believe I could learn SQL using this tool.*" Similarly, P24 stated, "*I wish SQLucid can be made available as a website. It can be used to teach beginners SQL knowledge and I believe they are willing to pay for it.*"

9.5 Limitation and Future Directions

There are several limitations in the design of our user study. First, although our participants represented a wide range of expertise levels in SQL, they were all university students. In the future, we plan to recruit industrial practitioners to study the real-world adoption and ecological validity of SQLucid. We will also conduct semi-structured interviews and surveys to gather feedback from industrial practitioners. Second, we did not explicitly measure user perception of accuracy, but user confidence is a useful proxy for it. Figure 8 shows a significant improvement in the confidence of SQLucid compared to DIY and MISP. Figure 11 shows each key feature in SQLucid contributes to increase user confidence.

The current design of SQLucid offers room for further improvement. First, to further enhance its educational potential, SQLucid can establish a triple-linkage among the SQL statement, SQL explanation, and corresponding database content. Combined with the intermediate query results, this can serve as a promising learning tool for SQL beginners to understand both the syntax and semantics of SQL queries. Furthermore, as suggested by several participants, SQLucid can benefit from displaying more information about the SQL generation process, such as model confidence scores. This additional information could direct users' attention and help them determine which steps of the query they should prioritize. Another future direction could focus on automatically reordering edited steps. SQLucid currently assumes users know exactly where to add new steps. Supporting automatic step reordering can eliminate this assumption.

10 CONCLUSION

This paper presented SQLucid, a novel interactive SQL refinement interface that enables users to effectively query data from relational databases using natural language. SQLucid integrates editable explanation, visual correspondence, intermediate query results, and other auxiliary features. These features echoed with each other, creating a grounded natural language interface with rich interactions for users to understand the generated queries, identify errors, and correct any errors. A user study with 30 participants shows that SQLucid can help users query data more quickly and accurately, with increased confidence and reduced cognitive load. A user study with 8 participants demonstrates the effectiveness of key features in SQLucid. A quantitative experiment with 100 query tasks indicates that SQLucid can be generalized to various tasks.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their helpful and detailed feedback, as well as the time and care they spent reviewing our work. We thank all the participants in the pilot study and two user studies for their valuable comments. This work was supported in part by Amazon Research Award and the National Science Foundation (NSF Grant ITE-2333736).

REFERENCES

[1] Serge Abiteboul, Rakesh Agrawal, Phil Bernstein, Mike Carey, Stefano Ceri, Bruce Croft, David DeWitt, Mike Franklin, Hector Garcia Molina, Dieter Gawlick, Jim Gray, Laura Haas, Alon Halevy, Joe Hellerstein, Yannis Ioannidis, Martin Kersten, Michael Pazzani, Mike Lesk, David Maier, Jeff Naughton, Hans Schek, Timos Sellis, Avi Silberschatz, Mike Stonebraker, Rick Snodgrass, Jeff Ullman, Gerhard Weikum, Jennifer Widom, and Stan Zdonik. 2005. The Lowell Database Research Self-Assessment. *Commun. ACM* 48, 5 (may 2005), 111–118. https://doi.org/10.1145/1060710.1060718

[2] Zahra Ashktorab, Mohit Jain, Q Vera Liao, and Justin D Weisz. 2019. Resilient chatbots: Repair strategy preferences for conversational breakdowns. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–12.

[3] Christopher Baik, H. V. Jagadish, and Yunyao Li. 2019. Bridging the Semantic Gap with SQL Query Logs in Natural Language Interfaces to Databases. *CoRR* abs/1902.00031 (2019). arXiv:1902.00031 http://arxiv.org/abs/1902.00031

[4] Gagan Bansal, Tongshuang Wu, Joyce Zhou, Raymond Fok, Besmira Nushi, Ece Kamar, Marco Tulio Ribeiro, and Daniel Weld. 2021. Does the whole exceed its parts? the effect of ai explanations on complementary team performance. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–16.

[5] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2022. Grounded Copilot: How Programmers Interact with Code-Generating Models. arXiv:2206.15000 [cs.HC]

[6] Erin Beneteau, Olivia K Richards, Mingrui Zhang, Julie A Kientz, Jason Yip, and Alexis Hiniker. 2019. Communication breakdowns between families and Alexa. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–13.

[7] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2023. Taking Flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools. *Queue* 20, 6 (jan 2023), 35–57. https://doi.org/10.1145/3582083

[8] HOCK C. CHAN, BERNARD C.Y. TAN, and KWOK-KEE WEI. 1999. Three Important Determinants of User Performance for Database Retrieval. *Int. J. Hum.-Comput. Stud.* 51, 5 (nov 1999), 895–918. https://doi.org/10.1006/ijhc.1999.0272

[9] Sarah E Chasins, Maria Mueller, and Rastislav Bodik. 2018. Rousillon: Scraping distributed hierarchical web data. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 963–975.

[10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[11] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[12] H. Clark and S. Brennan. 1991. Grounding in Communication', 127-149 in Resnick Lb, Levine Jm and Teasley Sd. In *Perspectives on Socially Shared Cognition*, Lauren Resnick, Levine B., M. John, Stephanie Teasley, and D. (Eds.). American Psychological Association, 259–292.

[13] Michelle Patrick Cook. 2006. Visual representations in science education: The influence of prior knowledge and cognitive load theory on instructional design principles. *Science Education* 90, 6 (2006), 1073–1091. https://doi.org/10.1002/sce.20164 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/sce.20164

[14] Felipe Costa, Sixun Ouyang, Peter Dolog, and Aonghus Lawlor. 2018. Automatic Generation of Natural Language Explanations. In *Proceedings of the 23rd International Conference on Intelligent User Interfaces Companion* (Tokyo, Japan) *(IUI '18 Companion)*. Association for Computing Machinery, New York, NY, USA, Article 57, 2 pages. https://doi.org/10.1145/3180308.3180366

[15] Ian Drosos, Titus Barik, Philip J Guo, Robert DeLine, and Sumit Gulwani. 2020. Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–12.

[16] Upol Ehsan, Brent Harrison, Larry Chan, and Mark O. Riedl. 2018. Rationalization: A Neural Machine Translation Approach to Generating Natural Language Explanations. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society* (New Orleans, LA, USA) *(AIES '18)*. Association for Computing Machinery, New York, NY, USA, 81–87. https://doi.org/10.1145/3278721.3278736

[17] Malin Eiband, Sarah Theres Völkel, Daniel Buschek, Sophia Cook, and Heinrich Hussmann. 2019. When people and algorithms meet: User-reported problems in intelligent everyday applications. In *Proceedings of the 24th international conference on intelligent user interfaces*. 96–106.

[18] Ahmed Elgohary, Saghar Hosseini, and Ahmed Hassan Awadallah. 2020. Speak to your Parser: Interactive Text-to-SQL with Natural Language Feedback. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 2065–2077. https://doi.org/10.18653/v1/2020.acl-main.187

[19] Ahmed Elgohary, Christopher Meek, Matthew Richardson, Adam Fourney, Gonzalo Ramos, and Ahmed Hassan Awadallah. 2021. NL-EDIT: Correcting Semantic Parse Errors through Natural Language Interaction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 5599–5610. https://doi.org/10.18653/v1/2021.naacl-main.444

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

[21] Shalini Ghosh, Giedrius Burachas, Arijit Ray, and Avi Ziskind. 2019. Generating Natural Language Explanations for Visual Question Answering using Scene Graphs and Visual Attention. *CoRR* abs/1902.05715 (2019). arXiv:1902.05715 http://arxiv.org/abs/1902.05715

[22] Barbara J. Grosz. 1983. TEAM: A Transportable Natural-Language Interface System. In *First Conference on Applied Natural Language Processing*. Association for Computational Linguistics, Santa Monica, California, USA, 39–45. https://doi.org/10.3115/974194.974201

[23] Sumit Gulwani, José Hernández-Orallo, Emanuel Kitzelmann, Stephen H Muggleton, Ute Schmid, and Benjamin Zorn. 2015. Inductive programming meets the real world. *Commun. ACM* 58, 11 (2015), 90–99.

[24] Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. DialSQL: Dialogue Based Structured Query Generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, 1339–1349. https://doi.org/10.18653/v1/P18-1124

[25] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Human Mental Workload*, Peter A. Hancock and Najmedin Meshkati (Eds.). Advances in Psychology, Vol. 52. North-Holland, 139–183. https://doi.org/10.1016/S0166-4115(08)62386-9

[26] Andrew Head, Codanda Appachu, Marti A. Hearst, and Björn Hartmann. 2015. Tutorons: Generating context-relevant, on-demand explanations and demonstrations of online code. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 3–12. https://doi.org/10.1109/VLHCC.2015.7356972

[27] Jingmei Hu, Priyan Vaithilingam, Stephen Chong, Margo Seltzer, and Elena L Glassman. 2021. Assuage: Assembly synthesis using a guided exploration. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 134–148.

[28] Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization. In *ArXiv preprint arXiv:1902.01069*. arXiv. https://doi.org/10.48550/ARXIV.1902.01069

[29] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, 963–973. https://doi.org/10.18653/v1/P17-1089

[30] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. 2007. Making Database Systems Usable. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data* (Beijing, China) *(SIGMOD '07)*. Association for Computing Machinery, New York, NY, USA, 13–24. https://doi.org/10.1145/1247480.1247483

[31] Dhanya Jayagopal, Justin Lubin, and Sarah E Chasins. 2022. Exploring the learnability of program synthesizers by novice programmers. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–15.

[32] Xue Jiang, Yihong Dong, Lecheng Wang, Qiwei Shang, and Ge Li. 2023. Self-planning code generation with large language model. *arXiv preprint arXiv:2303.06689* (2023).

[33] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the sigchi conference on human factors in computing systems*. 3363–3372.

[34] Rohit J Kate, Yuk Wah Wong, Raymond J Mooney, et al. 2005. Learning to transform natural to formal languages. In *AAAI*, Vol. 5. 1062–1068.

[35] Rafal Kocielnik, Saleema Amershi, and Paul N Bennett. 2019. Will you accept an imperfect ai? exploring designs for adjusting end-user expectations of ai systems. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–14.

[36] Andreas Kokkalis, Panagiotis Vagenas, Alexandros Zervakis, Alkis Simitsis, Georgia Koutrika, and Yannis Ioannidis. 2012. Logos: A System for Translating Queries into Narratives. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (Scottsdale, Arizona, USA) *(SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 673–676. https://doi.org/10.1145/2213836.2213929

[37] Georgia Koutrika, Alkis Simitsis, and Yannis E. Ioannidis. 2010. Explaining structured queries in natural language. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. 333–344. https://doi.org/10.1109/ICDE.2010.5447824

[38] Philippe Laban, Wojciech Kryściński, Divyansh Agarwal, Alexander R. Fabbri, Caiming Xiong, Shafiq Joty, and Chien-Sheng Wu. 2023. LLMs as Factual Reasoners: Insights from Existing Benchmarks and Beyond. arXiv:2305.14540 [cs.CL]

[39] Tak Yeon Lee, Casey Dugan, and Benjamin B Bederson. 2017. Towards understanding human mistakes of programming by example: an online user study. In *Proceedings of the 22Nd International Conference on Intelligent User Interfaces*. 257–261.

[40] Gilly Leshed, Eben M Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1719–1728.

[41] Aristotelis Leventidis, Jiahui Zhang, Cody Dunne, Wolfgang Gatterbauer, H.V. Jagadish, and Mirek Riedewald. 2020. QueryVis: Logic-Based Diagrams Help Users Understand Complicated SQL Queries Faster. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 2303–2318. https://doi.org/10.1145/3318464.3389767

[42] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proc. VLDB Endow.* 8, 1 (sep 2014), 73–84. https://doi.org/10.14778/2735461.2735468

[43] Fei Li and Hosagrahar V Jagadish. 2014. NaLIR: An Interactive Natural Language Interface for Querying Relational Databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 709–712. https://doi.org/10.1145/2588555.2594519

[44] Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 6038–6049. https://doi.org/10.1145/3025453.3025483

[45] Toby Jia-Jun Li, Marissa Radensky, Justin Jia, Kirielle Singarajah, Tom M. Mitchell, and Brad A. Myers. 2019. PUMICE: A Multi-Modal Agent that Learns Concepts and Conditionals from Natural Language and Demonstrations. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST 2019)*. ACM. https://doi.org/10.1145/3332165.3347899

[46] Yuntao Li, Bei Chen, Qian Liu, Yan Gao, Jian-Guang Lou, Yan Zhang, and Dongmei Zhang. 2020. "What Do You Mean by That?" A Parser-Independent Interactive Approach for Enhancing Text-to-SQL. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 6913–6922. https://doi.org/10.18653/v1/2020.emnlp-main.561

[47] Q Vera Liao, Daniel Gruen, and Sarah Miller. 2020. Questioning the AI: informing design practices for explainable AI user experiences. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–15.

[48] J. C. R. Licklider. 1960. Man-Computer Symbiosis. *IRE Transactions on Human Factors in Electronics* HFE-1, 1 (1960), 4–11. https://doi.org/10.1109/THFE2.1960.4503259

[49] Michael Xieyang Liu, Advait Sarkar, Carina Negreanu, Benjamin Zorn, Jack Williams, Neil Toronto, and Andrew D. Gordon. 2023. "What It Wants Me To Say": Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. ACM. https://doi.org/10.1145/3544548.3580817

[50] Ewa Luger and Abigail Sellen. 2016. " Like Having a Really Bad PA" The Gulf between User Expectation and Experience of Conversational Agents. In *Proceedings of the 2016 CHI conference on human factors in computing systems*. 5286–5297.

[51] Mikaël Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Alex Polozov, Rishabh Singh, Ben Zorn, and Sumit Gulwani. 2015. User Interaction Models for Disambiguation in Programming by Example. In *28th ACM User Interface Software and Technology Symposium (UIST 2015)* (28th acm user interface software and technology symposium (uist 2015) ed.). ACM – Association for Computing Machinery. https://www.microsoft.com/en-us/research/publication/user-interaction-models-for-disambiguation-in-programming-by-example/

[52] Daphne Miedema and George Fletcher. 2021. SQLVis: Visual Query Representations for Supporting SQL Learners. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–9. https://doi.org/10.1109/VL/HCC51201.2021.9576431

[53] Antonija Mitrovic. 1998. Learning SQL with a Computerized Tutor. *SIGCSE Bull.* 30, 1 (mar 1998), 307–311. https://doi.org/10.1145/274790.274318

[54] Arpit Narechania, Adam Fourney, Bongshin Lee, and Gonzalo Ramos. 2021. DIY: Assessing the Correctness of Natural Language to SQL Systems. In *26th International Conference on Intelligent User Interfaces* (College Station, TX, USA) *(IUI '21)*. Association for Computing Machinery, New York, NY, USA, 597–607. https://doi.org/10.1145/3397481.3450667

[55] Zheng Ning, Zheng Zhang, Tianyi Sun, Yuan Tian, Tianyi Zhang, and Toby Jia-Jun Li. 2023. An Empirical Study of Model Errors and User Error Discovery and Repair Strategies in Natural Language Database Queries. In *Proceedings of the 28th International Conference on Intelligent User Interfaces (IUI '23)*.

[56] Saswat Padhi, Todd Millstein, Aditya Nori, and Rahul Sharma. 2019. Overfitting in synthesis: Theory and practice. In *International Conference on Computer Aided Verification*. Springer, 315–334.

[57] Pruthvi Patel, Swaroop Mishra, Mihir Parmar, and Chitta Baral. 2022. Is a Question Decomposition Unit All We Need?. In *2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022*.

[58] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*. COLING, Geneva, Switzerland, 141–147. https://aclanthology.org/C04-1021

[59] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. arXiv:2304.11015 [cs.CL]

[60] Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive Bottom-up Semantic Parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 311–324. https://doi.org/10.18653/v1/2021.naacl-main.29

[61] Cynthia Rudin. 2018. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. (2018). https://doi.org/10.48550/ARXIV.1811.10154

[62] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. 2016. ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores. *Proc. VLDB Endow.* 9, 12 (aug 2016), 1209–1220. https://doi.org/10.14778/2994509.2994536

[63] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 9895–9901. https://doi.org/10.18653/v1/2021.emnlp-main.779

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

[65] Ben Shneiderman. 1983. Direct manipulation: A step beyond programming languages. *Computer* 16, 08 (1983), 57–69.

[66] Alkis Simitsis and Yannis Ioannidis. 2009. DBMSs Should Talk Back Too. In *10.48550/ARXIV.0909.1786*. arXiv. https://doi.org/10.48550/ARXIV.0909.1786

[67] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2022. Llm-planner: Few-shot grounded planning for embodied agents with large language models. *arXiv preprint arXiv:2212.04088* (2022).

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

[69] Yuan Tian, Zheng Zhang, Zheng Ning, Toby Jia-Jun Li, Jonathan K. Kummerfeld, and Tianyi Zhang. 2024. Interactive Text-to-SQL Generation via Editable Step-by-Step Explanations. arXiv:2305.07372 [cs.DB]

[70] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 332, 7 pages. https://doi.org/10.1145/3491101.3519665

[71] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*. 1–7.

[72] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 7567–7578. https://doi.org/10.18653/v1/2020.acl-main.677

[73] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv:2203.11171 [cs.CL]

[74] David H.D. Warren and Fernando C.N. Pereira. 1982. An Efficient Easily Adaptable System for Interpreting Natural Language Queries. *American Journal of Computational Linguistics* 8, 3-4 (1982), 110–122. https://aclanthology.org/J82-3002
[75] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *CoRR* abs/2201.11903 (2022). arXiv:2201.11903 https://arxiv.org/abs/2201.11903
[76] William Woods, Ronald Kaplan, and Bonnie Webber. 1972. The Lunar Science Natural Language Information System: Final Report. (01 1972).
[77] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, and Vadim Sheinin. 2018. SQL-to-Text Generation with Graph-to-Sequence Model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 931–936. https://doi.org/10.18653/v1/D18-1112
[78] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning. *CoRR* abs/1711.04436 (2017). arXiv:1711.04436 http://arxiv.org/abs/1711.04436
[79] Tianci Xue, Ziqi Wang, Zhenhailong Wang, Chi Han, Pengfei Yu, and Heng Ji. 2023. RCOT: Detecting and Rectifying Factual Inconsistency in Reasoning by Reversing Chain-of-Thought. arXiv:2305.11499 [cs.CL]
[80] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: Query Synthesis from Natural Language. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 63 (oct 2017), 26 pages. https://doi.org/10.1145/3133887
[81] Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019. Model-based Interactive Semantic Parsing: A Unified Framework and A Text-to-SQL Case Study. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 5447–5458. https://doi.org/10.18653/v1/D19-1547
[82] Kuat Yessenov, Shubham Tulsiani, Aditya Menon, Robert C Miller, Sumit Gulwani, Butler Lampson, and Adam Kalai. 2013. A colorful approach to text processing by example. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. 495–504.
[83] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 3911–3921. https://doi.org/10.18653/v1/D18-1425
[84] JD Zamfirescu-Pereira, Richmond Y Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–21.
[85] John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*. 1050–1055.
[86] Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 5338–5349. https://doi.org/10.18653/v1/D19-1537
[87] Tianyi Zhang, London Lowmanstone, Xinyu Wang, and Elena L Glassman. 2020. Interactive program synthesis by augmented examples. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 627–648.
[88] Zhanhui Zhou, Man To Tang, Qiping Pan, Shangyin Tan, Xinyu Wang, and Tianyi Zhang. 2022. INTENT: Interactive Tensor Transformation Synthesis. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–16.

A USER RATINGS OF INDIVIDUAL FEATURES

In the post-task survey, participants rated the usefulness of key features of SQLucid in 7-point Likert scale questions. Figure 13 summarizes the distribution of user ratings.

We found that the majority of participants were satisfied with each feature in SQLucid. The most appreciated features were *being able to understand the SQL query via the step-by-step explanation* and *being able to directly edit the explanation in natural language to fix an error*. P10 wrote, "*I really enjoyed this tool [SQLucid] a lot better than the previous two. Doing everything in natural language is way more direct. I don't have to answer strange questions or click confusing options [in drop-down menus]...By editing the steps, I was able to get more answers than previous tools.*"

Furthermore, 30 participants agreed or strongly agreed that "*seeing the intermediate execution results helps me understand the SQL query and validate its correctness.*" P23 commented, "*I liked how intermediate steps and results were shown so users could see how the system interpreted the query.*" 29 participants agreed or strongly agreed that "*seeing the highlighted tables/columns helps me understand the NL description.*" P5 wrote, "*the highlighting feature is useful for users to locate the corresponding elements quickly.*" Even the least appreciated feature—the edit history of SQL explanations—was still considered convenient by the majority of participants (25/30). P14 wrote, "*I also liked how easy it was to go in and edit the query as well as go back if I made a mistake.*"

B USER STUDY TASKS

Table 6 present examples of tasks with different difficulty levels from the 48 tasks used in our study. Table 6 also render the databases these tasks were operated on, as well as the ground-truth SQL queries for these tasks. These tasks were selected from the Spider benchmark [83]. Spider is a large-scale, complex, and cross-domain benchmark, consisting of databases with multiple tables. It has become the de facto standard for measuring text-to-SQL models these days. Spider categorizes these tasks into four difficulty levels—easy, medium, hard, and extra hard. We performed a stratified random sampling on the tasks from Spider [83]. Specifically, we selected 12 easy tasks, 12 medium tasks, 12 hard tasks, and 12 extra hard tasks, according to the difficulty classification from Spider. For each participant and each tool/variant assignment during the study, we randomly selected 2 tasks per difficulty level from the pool of 48 tasks, resulting in 8 tasks per condition. We randomized the order of the 8 tasks to counterbalance the impact of task difficulty levels (e.g., doing easy tasks first vs. doing difficult tasks first). If a participant found a task too difficult to solve, they were allowed to skip it.

C USER INTERFACES OF SQLUCID AND BASELINES

This section demonstrates the user interface (UI) of baseline tools used in our user study I.

MISP. Given a natural language question, MISP may ask users multiple-choice questions to clarify which column should be considered. If none of the listed choices are correct, users are allowed to provide their own answers. The user's answer is used to constrain the decoding process by adjusting the probability of code tokens induced by the answer. However, MISP directly renders the generated SQL to users without explanation. Therefore, users need to be familiar with SQL syntax to identify errors. The official implementation of MISP on GitHub only had a command-line interface, and the original text-to-SQL model [86] had much lower accuracy than newer models. To enable a fair comparison, we first created an interface for MISP, which includes everything from the SQLucid interface except the *Query Explanation* view (Figure 3 Ⓓ). Then, we replaced their text-to-SQL model [86] with the one [60] used in SQLucid. Thus, the only difference between the two systems is the interaction mechanism.

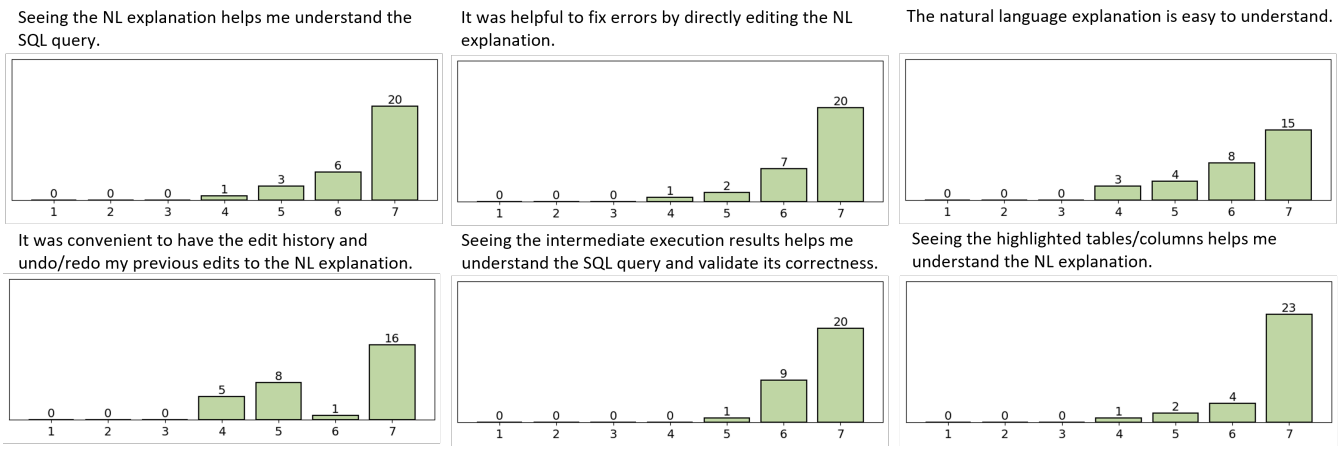

Figure 13: User ratings on individual features (1—strong disagreement, 7—strong agreement)

As shown in Figure 14, MISP shares a similar UI as SQLucid (Figure 3). For each query task, MISP allows users to select a database, inspect data in a table, and view the query result. The main difference from SQLucid is that MISP will render a generated query in the dialog and ask users to confirm whether the generated SQL is correct or not. If the user says the generated query is not correct, MISP will proactively predict the erroneous part and ask users to select alternative generations to fix the error. However, MISP does not provide a natural language explanation of the generated SQL. Users have to read and inspect the generated SQL in the dialog on their own, which is difficult for end-users who do not understand the syntax and semantics of SQL.

DIY. Given a natural language question, DIY creates a small sampled database and computes intermediate results on the samples. Furthermore, DIY maps tokens in a generated SQL query to words and phrases in the user-provided question. If the user finds an incorrect mapping (e.g., a wrong column name), they can fix it by selecting an alternative name and value from a drop-down menu. However, users cannot give further feedback in addition to selecting alternatives at certain locations. Since the original implementation of DIY is not publicly available, we reused the replication of DIY from Ning et al. [55] and designed a user interface similar to SQLucid. We also changed the original text-to-SQL model in Ning et al.'s implementation to the same model [60] of SQLucid for a fair comparison.

Figure 15 shows the UI of DIY. DIY only samples a small amount of data from a user-selected database to reduce the information overload of inspecting a large database. Users can type in a natural language question and then DIY generates a SQL query by invoking the base SQL generation model. DIY automatically matches tokens in the natural language question with tokens in the generated SQL. Each matched natural language token is augmented with a drop-down menu with alternative SQL tokens predicted by the base model. If the prediction of a token is wrong, users can click on the drop-down menu and select an alternative token to fix it. Users can examine the query result, as well as the execution steps, in the bottom right view.

Table 6: Some example tasks in the user study

	Task	Ground truth SQL query
Easy	List the name of teachers whose hometown is not "Little Lever Urban District. **(course_teach)**	SELECT name FROM teacher WHERE hometown != "little lever urban district"
	What is the abbreviation for airline "JetBlue Airways" ? **(flight_2)**	SELECT Abbreviation FROM AIRLINES WHERE Airline = "JetBlue Airways"
	List all the student details in reversed lexicographical order. **(student_transcripts_tracking)**	SELECT other_student_details FROM Students ORDER BY other_student_details DESC
Medium	Which airlines have less than 200 flights? **(flights_2)**	SELECT T1.Airline FROM AIRLINES AS T1 JOIN FLIGHTS AS T2 ON T1.uid = T2.Airline GROUP BY T1.Airline HAVING COUNT(*) 200
	Who is the earliest graduate of the school? List the first name, middle name, and last name. **(flights_2)**	SELECT first_name , middle_name , last_name FROM Students ORDER BY date_left ASC LIMIT 1
	What are the countries having at least one car maker? List name and id. **(car_1)**	SELECT T1.CountryName , T1.CountryId FROM COUNTRIES AS T1 JOIN CAR_MAKERS AS T2 ON T1.CountryId = T2.Country GROUP BY T1.CountryId HAVING COUNT(*) = 1
Hard	What are the ids and names of the battles that led to more than 10 people killed in total? **(battle_death)**	SELECT T1.id , T1.name FROM battle AS T1 JOIN ship AS T2 ON T1.id = T2.lost_in_battle JOIN death AS T3 ON T2.id = T3.caused_by_ship_id GROUP BY T1.id HAVING SUM(T3.killed) 10
	What is the maximum number of times that a course shows up in different transcripts and what is that course's enrollment id? **(student_transcripts_tracking)**	SELECT COUNT(*) , student_course_id FROM Transcript_Contents GROUP BY student_course_id ORDER BY COUNT(*) DESC LIMIT 1
	What are the first names of the students who live in Haiti permanently or have the cell phone number 09700166582? **(student_transcripts_tracking)**	SELECT T1.first_name FROM students AS T1 JOIN addresses AS t2 ON T1.permanent_address_id = T2.address_id WHERE T2.country = 'haiti' OR T1.cell_mobile_number = '09700166582'
Extra hard	Which owner has paid the largest amount of money in total for their dogs? Show the owner id and zip code. **(dog_kennels)**	SELECT T1.owner_id , T1.zip_code FROM Owners AS T1 JOIN Dogs AS T2 ON T1.owner_id = T2.owner_id JOIN Treatments AS T3 ON T2.dog_id = T3.dog_id GROUP BY T1.owner_id ORDER BY sum(T3.cost_of_treatment) DESC LIMIT 1
	What is the area code in which the most voters voted? **(voter_1)**	SELECT T1.area_code FROM area_code_state AS T1 JOIN votes AS T2 ON T1.state = T2.state GROUP BY T1.area_code ORDER BY COUNT(*) DESC LIMIT 1
	What is the maximum horsepower and the make of the car models with 3 cylinders? **(car_1)**	SELECT T2.horsepower , T1.Make FROM CAR_NAMES AS T1 JOIN CARS_DATA AS T2 ON T1.MakeId = T2.Id WHERE T2.cylinders = 3 ORDER BY T2.horsepower DESC LIMIT 1

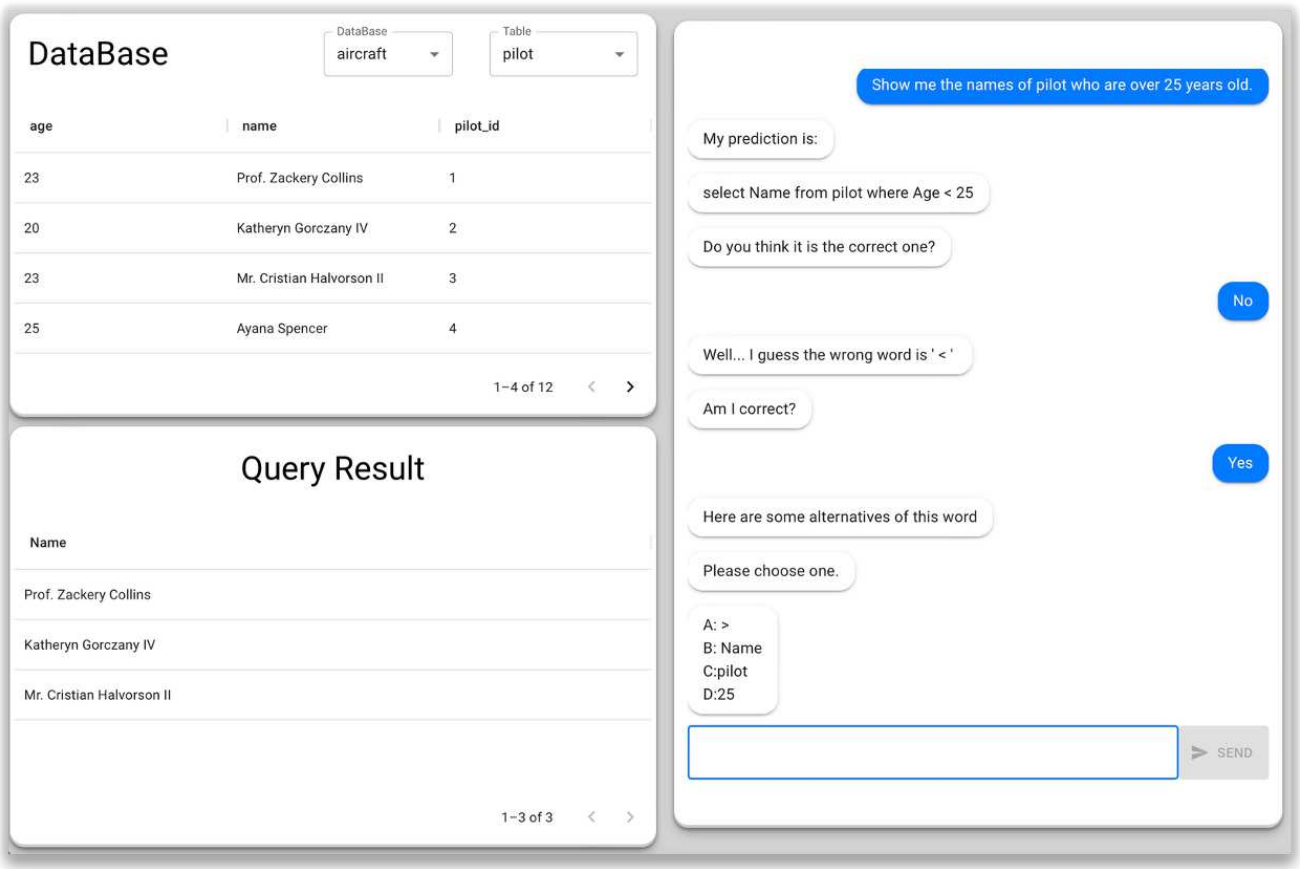

Figure 14: The UI of MISP

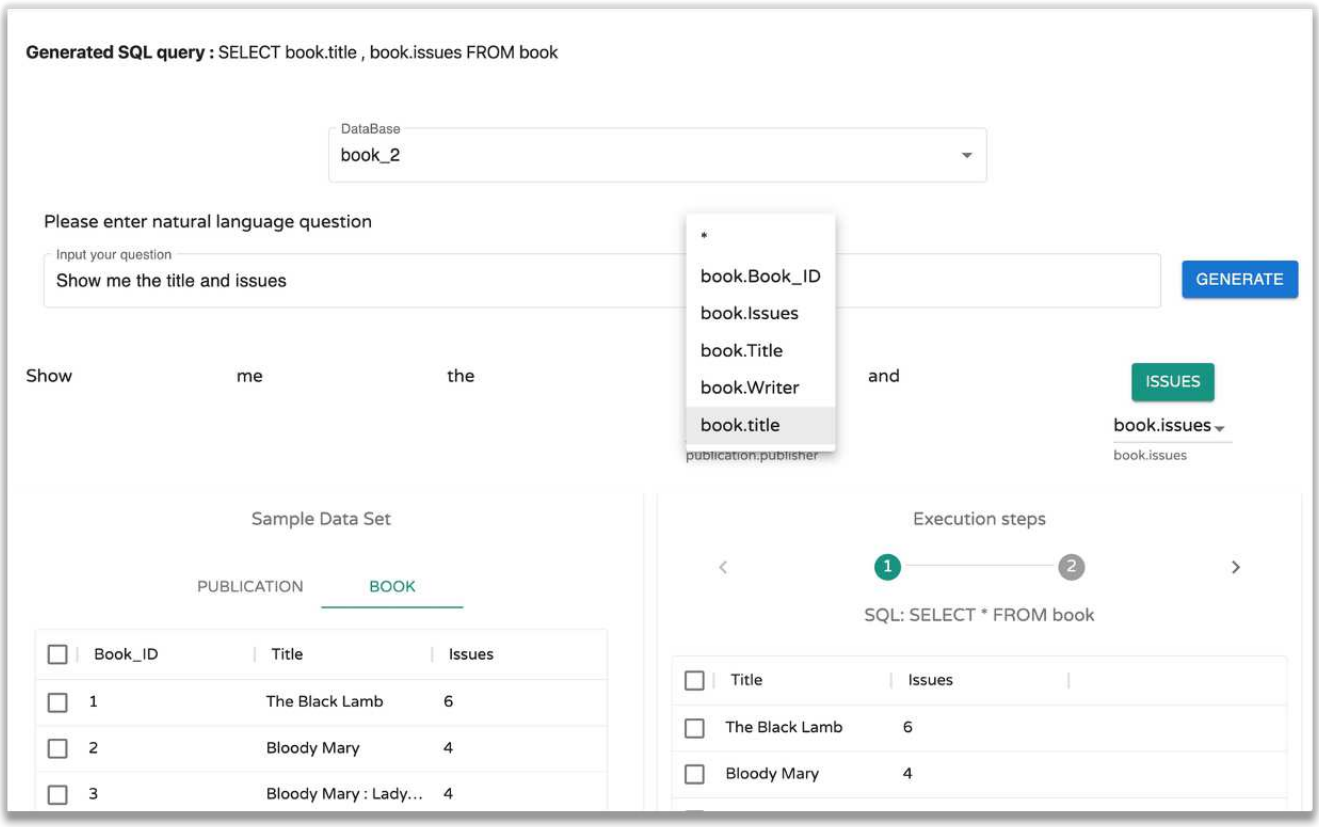

Figure 15: The UI of DIY