Interactive Text-to-SQL Generation via Editable Step-by-Step Explanations

Yuan Tian[1], **Zheng Zhang**[2], **Zheng Ning**[2],
Toby Jia-Jun Li[2], **Jonathan K. Kummerfeld**[3], **and Tianyi Zhang**[1]
Purdue University[1], University of Notre Dame[2], The University of Sydney[3]
tian211@purdue.edu, zzhang37@nd.edu, zning@nd.edu,
toby.j.li@nd.edu, jonathan.kummerfeld@sydney.edu.au, tianyi@purdue.edu

Abstract

Relational databases play an important role in this Big Data era. However, it is challenging for non-experts to fully unleash the analytical power of relational databases, since they are not familiar with database languages such as SQL. Many techniques have been proposed to automatically generate SQL from natural language, but they suffer from two issues: (1) they still make many mistakes, particularly for complex queries, and (2) they do not provide a flexible way for non-expert users to validate and refine the incorrect queries. To address these issues, we introduce a new interaction mechanism that allows users directly edit a step-by-step explanation of an incorrect SQL to fix SQL errors. Experiments on the Spider benchmark show that our approach outperforms three SOTA approaches by at least 31.6% in terms of execution accuracy. A user study with 24 participants further shows that our approach helped users solve significantly more SQL tasks with less time and higher confidence, demonstrating its potential to expand access to databases, particularly for non-experts.

1 Introduction

Natural language interfaces allow users to query a database via natural language. It significantly lowers the barrier to accessing databases, particularly for those without knowledge of specialized querying languages such as SQL. So far, a range of systems have been proposed to translate natural language queries to their corresponding SQL queries (Rubin and Berant, 2021; Wang et al., 2020; Scholak et al., 2021; Yu et al., 2020; Hwang et al., 2019). With the recent advancement in deep learning, the state-of-the-art text-to-SQL models have reached 79.9% execution accuracy on the Spider benchmark (Yu et al., 2018).

However, the rate of improvement has slowed, with a gain of only 5% since mid-2021. This is partly due to the inherent ambiguity of natural language and the complex structure of SQL queries

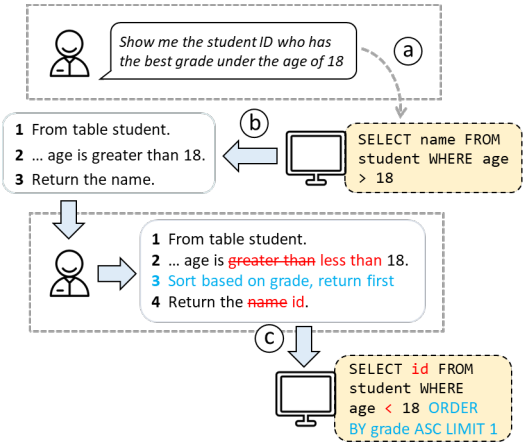

Figure 1: Refining a SQL query by directly editing a step-by-step explanation.

(e.g., nested or joined queries). Thus, it is challenging to generate a fully correct query in one step, especially for complex tasks (Yao et al., 2019).

There has been growing interest in developing "human-in-the-loop" approaches that elicit user feedback to guide SQL generation.

However, most of these approaches only support feedback in constrained forms, such as answering multiple-choice questions (MISP, PIIA, DialSQL Yao et al., 2019; Li et al., 2020; Gur et al., 2018) or changing incorrect keywords in a drop-down menu (DIY, Narechania et al., 2021). Such constrained feedback is not sufficient to fix many complex errors in real-world SQL tasks.

One exception is NL-EDIT (Elgohary et al., 2021), which allows users to provide feedback in an entirely new utterance. However, since the feedback is completely open-ended, it encounters similar NL understanding challenges as end-to-end models.

In this paper, we seek to strike a balance between constrained feedback and open-ended feedback by proposing a new interaction mechanism through editable step-by-step explanations. Fig. 1 illustrates our idea. This mechanism consists of three core components: (a) a text-to-SQL model, (b) an

explanation generation method, and (c) an SQL correction model.

Our key insight is that using a step-by-step explanation as the basis to suggest fixes allows users to precisely specify where the error is and how to fix it via direct edits. This not only saves users' time but also makes it easier for the model to locate the error and apply fixes.

Based on this idea, we implemented an interactive SQL generation and refinement system called STEPS. STEPS adopts a grammar-based method to generate step-by-step explanations and uses a hybrid method to convert a user-corrected explanation back to a SQL query. An evaluation with a simulated user on Spider (Yu et al., 2018) shows that STEPS can achieve 97.9% exact set matching accuracy, outperforming MISP, DIY, and NL-EDIT—by 33.5%, 33.2%, and 31.3% respectively.

To measure STEPS's performance with users, we conducted a user study with 24 real users.

We found that within the same amount of time, STEPS helped users complete almost 2X and 4X more tasks correctly than DIY and MISP, with significantly higher self-reported confidence and less mental load.[1]

This work makes the following contributions: (1) we propose a new interaction mechanism for the text-to-SQL task; (2) we develop an interactive text-to-SQL system based on the new interaction mechanism; (3) we conduct a comprehensive evaluation with both simulated and real users and demonstrate its effectiveness over state-of-the-art interactive systems. We will make our dataset and code publicly available upon acceptance.

2 Related Work

2.1 Text-to-SQL Generation

Natural language interfaces have long been recognized as a way to expand access to databases (Hendrix et al., 1978; Woods et al., 1972). The construction of several large text-to-SQL datasets, such as WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018), has enabled the adoption of deep learning models in this task, achieving unprecedented performance in recent years (Rubin and Berant, 2021; Wang et al., 2020; Scholak et al., 2021; Yu et al., 2020; Hwang et al., 2019). Our technique is based on the recent success of neural text-to-SQL models. Unlike existing models that perform end-to-end SQL generation, we propose a new interaction mechanism for users to validate and refine generated queries through step-by-step explanations.

As the first step to demonstrate the feasibility of our approach, we focus on single-turn SQL generation (Yu et al., 2018) in this work. There has also been recent work that supports multi-turn SQL generation (Yu et al., 2019a,b; Guo et al., 2021), where a sequence of interdependent queries are expressed in multiple utterances in a dialog. Models designed for multi-turn SQL generation typically need to reason about the dialog context and effectively encode the historical queries (Wang et al., 2021b; Hui et al., 2021; Zhang et al., 2019; Cai and Wan, 2020; Wang et al., 2021a). Our approach can be extended to support multi-turn SQL generation by initiating separate refinement sessions for individual queries while incorporating the contextual information of previous queries into explanation generation and text-to-clause generation.

2.2 Interactive Semantic Parsing for SQL

Recently, there has been a growing interest in interactive approaches that elicit user feedback to guide SQL generation. Iyer et al. (2017) deploys their model online and allows users to flag incorrect queries, based on which their model can be improved over time2. Both DIY (Narechania et al., 2021) and NaLIR (Li and Jagadish, 2014a,b) enable users to select alternative values or subexpressions to correct SQL. PIIA (Li et al., 2020), MISP (Yao et al., 2019), and DialSQL (Gur et al., 2018) proactively ask users to clarify their intent via multiple-choice questions.

Despite the great effort, existing approaches only solicit feedback in constrained forms, hindering their flexibility and effectiveness in addressing the variability of SQL errors. In contrast, our approach allows users to provide more open-ended feedback by directly editing the NL explanations generated by the model to specify what is wrong and how to fix it.

To the best of our knowledge, the only work that supports open-ended user feedback in SQL generation is NL-EDIT (Elgohary et al., 2021). NL-EDIT is trained on SPLASH (Elgohary et al., 2020), a dataset of SQL errors and user feedback utterance.

Given an incorrect SQL, NL-EDIT allows users

[1]We worked with the authors of NL-EDIT to include their system in the user study, but were unable to get it working due to missing code and other runtime errors. We use the accuracy reported in the NL-EDIT paper in the experiment.

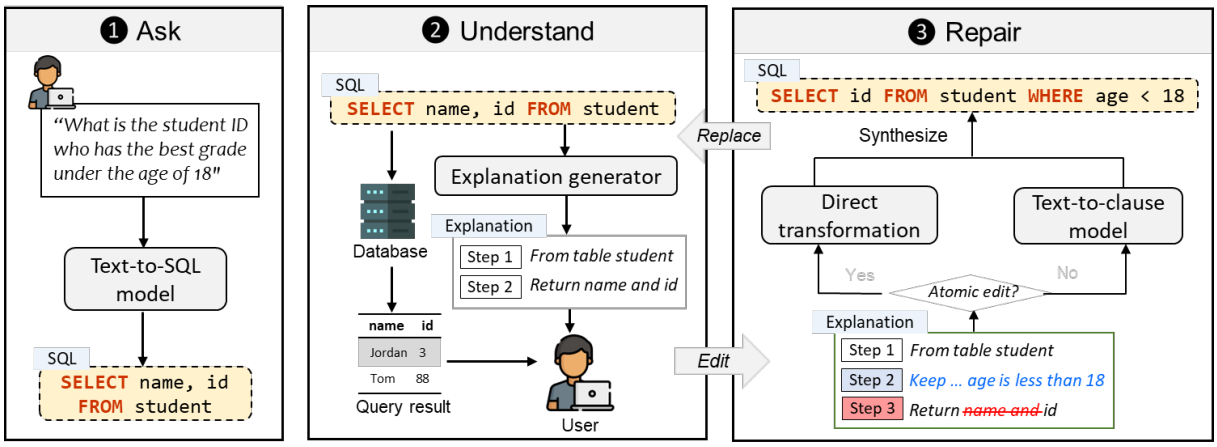

Figure 2: An Overview of Interactive SQL Generation and Refinement with Editable Step-by-Step Explanations

to provide a clarification utterance. Based on the utterance, the model generates a sequence of edits to the SQL query. Providing feedback via a new utterance introduces two challenges: (1) the model needs to infer which part of the SQL query to fix; (2) the model needs to predict what changes to apply. This increases the difficulty in interpreting user feedback, as a user can provide feedback in arbitrary ways. In contrast, our approach asks users to directly edit an NL explanation and make corrections to the explanation. Comparing the initial explanation with the user-corrected explanation makes it easier to locate the part of a SQL query that needs to be changed and infer what change to make.

The idea of SQL decomposition is similar to a recent work (Mo et al., 2022) that decomposes a user question to sub-questions on SPARQL. Their question decomposition model is trained on a crowd-sourced dataset. They further develop models that correct the initial parse by modifying the sub-questions using the correction templates (e.g., replace question #X with Y). In contrast, our grammar-based approach can generate human-readable, step-by-step explanations without the need for training a model. The explanation generated by the grammar empowers our system to map each entity in the explanation and SQL clause, paving the way for more efficient interaction mechanisms, as elaborated in Sec. 3.2.

2.3 Explaining SQL Queries in NL

Our approach is also related to prior work that generates NL explanations for SQL queries.

Simitsis and Ioannidis (2009) argued that databases should "talk back" in the human language so that users can verify the results. Kokkalis et al. (2012) and Koutrika et al. (2010) used a graph-based SQL translation approach, where each SQL is represented as a graph and the explanation is generated by traversing the graph. Elgohary et al. (2021, 2020) employed a template-based explanation approach, where they manually curated 57 templates for explanation generation. These existing approaches have limited capability to handle arbitrary SQL queries. To address this limitation, we propose a grammar-based method to first explain terminal tokens (e.g., operators, keywords) and gradually compose them into a complete explanation based on the derivation rules in the SQL grammar. Furthermore, none of the existing approaches supports *editable* explanations for SQL correction, which is a key feature of our approach.

3 Approach

Fig. 2 provides an overview of STEPS. Given a natural language (NL) question, STEPS invokes a text-to-SQL model to generate an initial SQL query. Then, it decomposes the generated SQL query into individual query clauses and re-orders them based on their execution order. Each clause is then translated into an NL description of the underlying data operation, which is then used to form a step-by-step explanation. By reading the NL explanation along with the query result, users can easily understand the behavior of the generated query and locate any errors, even if they are unfamiliar with SQL. If one step is incorrect, users can directly edit its explanation to specify the correct behavior. STEPS will then regenerate the clause based on the user-corrected explanation and substitute the original clause to fix the SQL query, rather than regenerating the entire query from scratch.

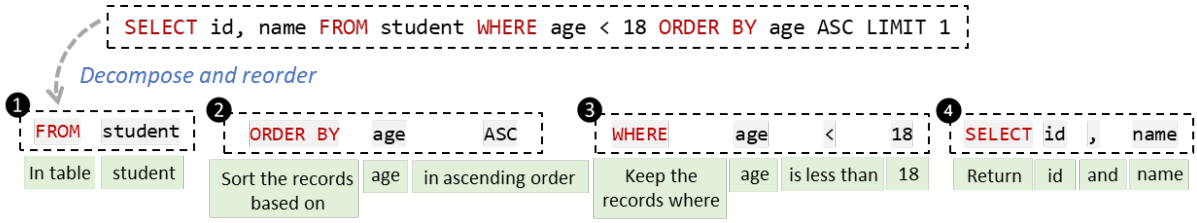

Figure 3: An example of the explanation generation process

3.1 Grammar-based SQL Explanation

To generate explanations for arbitrarily complex SQL queries (e.g., a query with nested subqueries), we design a grammar-based method to first decompose a query into individual clauses. Specifically, STEPS first parses a SQL query to its abstract syntax tree (AST) based on the SQL grammar in Table 5. Then, it traverses the AST to identify the subtree of each clause while preserving their hierarchical relations.

Given the subtree of a clause, STEPS performs an in-order traversal and translates each leaf node (i.e., a terminal token in the grammar) to the corresponding NL description based on the translation rules in Table 6. For example, `SELECT` is translated to "Return", while `Order By` is translated to "Sort the records based on". Then, STEPS concatenates these descriptions to form a complete sentence as the explanation of the clause.

Since SQL engines follow a specific order to execute individual clauses in a query[2], STEPS further reorders the clause explanations accordingly to reflect their execution order. We believe that this is a more faithful representation of the query behavior and thus can help users better understand the underlying data operations, compared with rendering them based on the syntactic order of clauses. Fig. 3 shows an example translation.

3.2 Text-to-Clause Generation

Given the user correction to a generated explanation, STEPS adopts a hybrid method to regenerate the corresponding SQL clause. For simple edits, such as replacing a column name, STEPS directly edits the original clause to fix the error based on three SQL transformation rules. For more complex edits, STEPS uses a neural text-to-clause model to regenerate the clause based on the user-corrected explanation.

The hybrid method is inspired by our observation that a large portion of SQL generation errors are simple errors (e.g., incorrect column names and operators), which can be fixed with a small edit. Thus, it is not necessary to regenerate the entire clause to fix such errors. Furthermore, compared to using a large model, direct transformation is more computationally efficient. Our experiment shows that direct transformation is 22K times faster than the text-to-clause model (Table 3).

3.2.1 Direct Transformation

We define three types of *atomic edits* that can be directly converted into SQL edits by STEPS: (1) replacing a column name, a table name, or a literal value (i.e., string, number), (2) adding a new column name in the explanation of a `SELECT` clause, and (3) removing a column name. While one can always design new transformation rules to support other simple edits, here we only focus on these three basic edit types in order to demonstrate the benefits of direct transformation.

The direct transformation algorithm works as follows. First, STEPS performs chunking on the original explanation e_o and the user-corrected explanation e_n. We choose to split an explanation into phrases rather than individual words, since it is more accurate to recognize column names and table names that are represented as compound nouns in an explanation. Then, the chunks are aligned using the Needleman and Wunsch (1970) algorithm. If a chunk from the original explanation is aligned with a chunk in the new explanation and both of them can be mapped to a column name, a table name, or a literal value, then STEPS replaces the corresponding name/value from the original clause with the new name/value. If a chunk from the new explanation is aligned with nothing, the chunk can be mapped to a column name, and the original clause is a `SELECT` clause, then STEPS directly appends the corresponding column name to the clause after a comma. If a chunk from the old explanation is aligned with nothing and the chunk can be mapped to a column name, then STEPS directly removes the corresponding column name from the clause.

[2] https://sqlbolt.com/lesson/select_queries_order_of_execution

3.2.2 Text-to-Clause Model

For more complex edits, we develop a text-to-clause model. We adopt the model architecture of SmBoP (Rubin and Berant, 2021) for this model. SmBoP is a semi-autoregressive and bottom-up transformer-based semantic parser for SQL. It decodes subtrees first and then gradually combines them to form a complete AST of the final SQL. To train the text-to-clause model, we automatically created a dataset with 83K text-clause pairs based on Spider (Yu et al., 2018). Specifically, for each SQL query in Spider, we use the explanation generation method in Section 3.1 to decompose them into clauses and generate an NL explanation of each clause. To improve the diversity of NL explanations, we paraphrase the original explanations in two ways. First, we design a set of rules to randomly replace words with their synonyms, as shown in Table 8. Second, we paraphrase the explanation using an online paraphrasing tool called QuillBot[3]. Since QuillBot does not provide an API, we develop a web automation script to automate this process using PyAutoGUI[4]. We train the text-to-clause model using the Adam Optimizer with a learning rate of $1.8e-4$ and a dropout rate of 0.1. We perform a 10-fold validation and the exact set matching accuracy of our model is 90.6%. More details can be found in Appendix D.

3.3 SQL Rewriting and Composition

After regenerating the clauses for all user-corrected explanations, STEPS composes them together to form a new query while reconciling possible syntax errors based on several rewriting rules.

Simply combining SQL clauses may lead to syntactic errors. As shown in Fig. 4, the regenerated clause may reference another table that does not exist in the previous query, e.g., `info` in the second clause. Thus, we design several rewriting rules to fix such errors. First, if a table is referenced but is not the table in the `FROM` clause, STEPS rewrites the `FROM` clause to join the existing table with the new table based on the foreign key. Second, if multiple `SELECT`, `WHERE`, or `HAVING` clauses are at the same hierarchical level, STEPS merges them into a single clause. Third, if there are multiple `ORDER BY` or `GROUP BY` clauses, STEPS only keeps the first one. Fig. 4 shows an example of the rewriting process.

[3] https://quillbot.com

[4] https://pypi.org/project/PyAutoGUI

4 Experiment

To evaluate the performance of STEPS, we conducted quantitative experiments on the Spider benchmark (Yu et al., 2018) with three SOTA interactive SQL generation approaches—MISP (Yao et al., 2019), DIY (Narechania et al., 2021), and NL-EDIT (Elgohary et al., 2021).

4.1 User Simulation & Setup

In order to perform a quantitative evaluation of STEPS on the Spider benchmark (Yu et al., 2018), we developed an automated script to simulate user feedback following the user simulation in Yao et al. (2019). Specifically, given a generated query and the ground-truth query, our script decomposes both of them into clauses using the method described in Section 3.1. Then, it compares the clauses and checks their semantic equivalence using the component matching method of Yu et al. (2018). For example, `SELECT name, age` is considered semantically equivalent to `SELECT age, name`.

The simulated user provides feedback when the generated query is not semantically equivalent to the corresponding clause in the ground truth (i.e., there is an error). There are three cases. First, if the generated query contains a clause that does not exist in the ground truth, our script will delete its explanation from the original explanation. Second, if the generated query does not contain a clause from the ground truth, our script will generate the NL explanation of this missing clause using the explanation generation method described in Section 3.1, paraphrase it using QuillBot, and insert it into the corresponding location of the original explanation. Finally, if the generated query contains an inconsistent clause from the ground truth, our script will generate the NL explanation based on the correct clause in the ground truth, paraphrase it using QuillBot, and replace the explanation of the incorrect clause with the paraphrased one.

4.2 Comparsion Baselines

We compared STEPS to three state-of-the-art interactive SQL generation methods:

MISP (Yao et al., 2019) enables users to give feedback by answering multiple-choice questions. For example, MISP may ask users to clarify whether a column should be considered in the query, and then the user can answer yes or no to give feedback. The user's answer is then used to constrain the decoding process by adjusting the

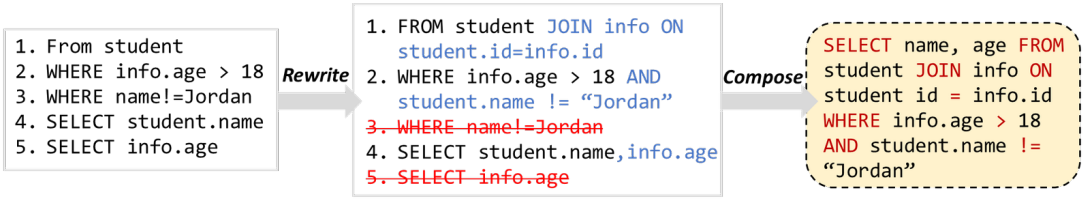

Figure 4: An example of SQL clause rewriting and composition

probability of code tokens induced by the answer. We used the original implementation of MISP from their GitHub repository.

DIY (Narechania et al., 2021) enables users to refine a generated SQL using a drop-down menu over the original NL question to select alternative table names, column names, operators, and aggregate functions. We reimplemented DIY since no open-source implementation is available. Specifically, to construct the word-entity mapping, we calculate word embedding semantic similarity. In the user simulation, we align the generated SQL with the ground truth SQL. If an entity in the generated SQL is not present in the ground truth SQL, which indicates an error, and it has been mapped to the NL question, which means users can give feedback via a drop-down menu, we replace it with the corresponding ground truth entity.

NL-EDIT (Elgohary et al., 2021) enables users to correct errors by telling the system how to modify the SQL in NL. User feedback will be parsed into a set of simple edits (e.g., add, remove) that are applied to the initial generation. We report results for NL-EDIT using the accuracy numbers from the NL-EDIT paper. We worked with the NL-EDIT authors to try to run the system, but were unable to resolve issues due to missing code and other run-time errors.

4.3 Results

Table 1 shows the exact set matching accuracy comparison between STEPS and the baselines. Following the experimental design of MISP and NL-EDIT, we use EditSQL (Zhang et al., 2019) as the base SQL generation model and exact set matching accuracy (Yu et al., 2018) as the evaluation metric. STEPS achieves 97.9% accuracy, outperforming all three baselines by at least 31%. This result shows that allowing users to specify the exact erroneous steps and only regenerating the incorrect clauses rather than the entire query can accurately fix most of the SQL generation errors on the Spider benchmark.

	$\textbf{Acc}_{\text{set}}$
EditSQL (Zhang et al., 2019)	0.576
+ MISP (Yao et al., 2019)	0.644
+ DIY (Narechania et al., 2021)	0.647
+ NL-EDIT (Elgohary et al., 2021)	0.666
+ STEPS	**0.979**

Table 1: Exact Set Matching Accuracy Comparison

To demonstrate STEPS's performance is generalizable to other base models, we also evaluate STEPS on another model called SmBoP (Rubin and Berant, 2021). SmBoP is one of the best models on the Spider leaderboard with 74.5% exact set matching accuracy. Table 2 shows STEPS's exact set matching accuracy with SmBoP as the base model in comparison to EditSQL. We also report execution accuracy, another popular metric that compares the query results between the generated query and the ground truth. Note that since EditSQL does not predict any value in SQL conditions, the queries generated by EditSQL are not runnable. Thus, we cannot measure the execution accuracy of EditSQL. The result shows that STEPS consistently improves the accuracy of both models on SQL tasks with different levels of difficulty.[5] Specifically, STEPS can almost solve all easy and medium tasks and also achieves more than 90% accuracy for the hard and extra hard tasks.

Table 3 shows the ablation results of the hybrid method of STEPS. Regarding SQL generation accuracy, STEPS achieves comparable accuracy when using text-to-clause alone, while experiencing a significant accuracy degradation when using direct transformation alone. This makes sense since the direct transformation method is only designed to fix a small subset of the possible error types. However, for the types for which it is intended, the direct transformation approach is very accurate. As a result, using it as part of the hybrid system does not decrease accuracy, but does increase the efficiency.

[5]Spider categorizes their SQL tasks into four difficulty levels—easy, medium, hard, and extra hard.

	Acc_{set}					Acc_{exec}				
	Easy	**Medium**	**Hard**	**Extra hard**	**All**	**Easy**	**Medium**	**Hard**	**Extra hard**	**All**
EditSQL	0.681	0.632	0.456	0.395	0.576	-	-	-	-	-
+ STEPS	0.991	1.000	0.976	0.912	0.979	0.991	0.995	0.939	0.912	0.971
SmBoP	0.883	0.791	0.655	0.512	0.745	0.718	0.669	0.672	0.518	0.657
+ STEPS	0.992	1.000	0.977	0.916	0.981	0.992	0.995	0.943	0.916	0.973

Table 2: STEPS's Accuracy on SQL Tasks with Different Levels of Difficulty

	Acc_{set}	Acc_{exec}	**Time (ms)**
Direct transform only	0.788	0.745	0.0042
Text-to-clause only	0.981	0.973	95.53
Hybrid	0.981	0.973	57.24

Table 3: Ablation Study of the Hybrid Method

5 User Study

In addition to the quantitative experiments, we also conducted a user study with 24 real users to evaluate STEPS.[6]

5.1 Participants

We recruited 24 participants (22M, 2F) through mailing lists in an R1 university. To investigate how user expertise affects the performance of STEPS, participants were selected based on their familiarity with SQL. Specifically, 10 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 the syntax details when writing a SQL query (novice); 4 can fluently write SQL queries (expert). We shared the consent form with each participant and obtained their consent before each study. Each participant was compensated with a $25 gift card.

5.2 Comparison Baselines

We used MISP (Yao et al., 2019) and DIY (Narechania et al., 2021) as comparison baselines. As explained in Section 4.2, we did not use NL-EDIT, since we were unable to reproduce it. To ensure a fair comparison, we developed user interfaces with the same visual style for STEPS, MISP (Yao et al., 2019), and DIY (Narechania et al., 2021). The UI screenshots are provided in Appendix E.

5.3 SQL Tasks & Procedures

Each study includes 3 sessions, one for each tool. In each session, participants were asked to use the assigned tool to complete 8 SQL tasks in 10 minutes. We choose such an assignment based on 4 pilot studies before the user study. We found 10 minutes strikes a good balance between making participants patient and allowing enough time to complete tasks. To select the tasks, we first performed stratified random sampling on Spider to create a task pool of 24 SQL tasks, including 6 easy tasks, 6 medium tasks, 6 hard tasks, and 6 extra hard tasks. Before each session, we selected 2 tasks from each difficulty level from the task pool, which constitutes a total of 8 tasks to be solved in the session. To mitigate learning effects, the orders of both task assignment and tool assignment order were counterbalanced across participants.

Each session starts with participants watching a tutorial video of the assigned tool (6 min for STEPS, 3 min for MISP, and 2 min for DIY). Participants were given 5 minutes to practice and get familiar with the tool before working on real tasks. For each task, participants were asked to read the description of the task and then ask an initial NL question to the assigned tool. After receiving the generated query along with the query result, the participant can further validate and repair the generated query using the interaction mechanisms provided by the tool. Participants were allowed to skip a task if they found it too hard to solve.

At the end of each session, participants were asked to complete a post-task survey to rate their confidence about the final SQL query, how successful they perceived themselves in completing the tasks, and the mental effort to complete the tasks on a 7-point Likert scale. After all three sessions, participants completed a final survey, in which they directly compared the three tools. We recorded each study with the permission of the participants. Each study took an average of 79 minutes.

During each study, we asked participants to try their best to guide the SQL generation model to generate the correct query, rather than manually writing the query themselves. This is because some expert users may easily come up with a SQL query for easy or medium-level tasks without interacting with the SQL generation model, while our goal is to evaluate the effectiveness of the interaction.

[6]Our study was approved by our institution's IRB.

	Complete	Correct	Acc.	Skipped
MISP	3.0	1.7	0.57	1.4
DIY	5.4	3.5	0.68	0.8
STEPS	**6.7**↑	**5.7**↑	**0.86**↑	**0.3**↓

Table 4: User Performance (best results in bold). For all metrics, an ANOVA test indicated statistically significant mean differences across 3 tools (p-value < 0.01).

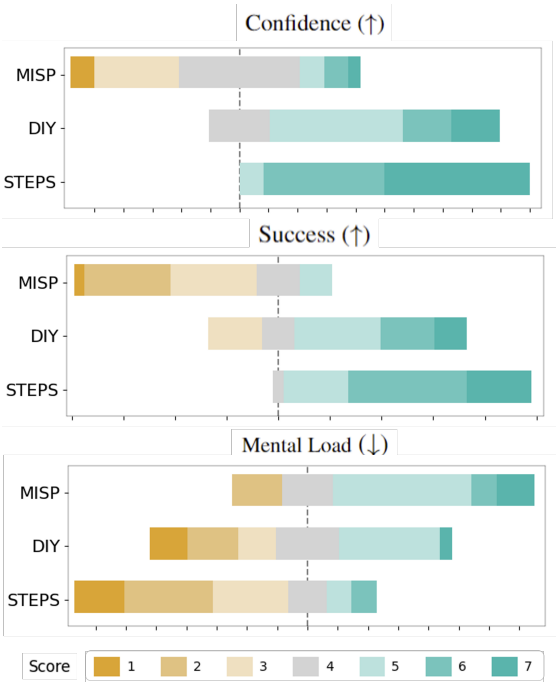

Figure 5: User Perception.

5.4 Results

Table 4 shows the average number of completed tasks, correct completions, task completion accuracy (#correct / #completed), and skipped tasks. We found that participants using STEPS completed more tasks compared to those using MISP and DIY. Furthermore, participants using STEPS completed significantly more tasks correctly than DIY and MISP, achieving the highest accuracy (85.81%) in SQL generation. Participants using STEPS barely skipped a task, implying that STEPS provides sufficient support for users to tackle challenging tasks so that users did not give up quickly. The ANOVA test shows that the mean differences in Table 4 are statistically significant among the three different conditions (p-value < 0.01). These results indicate that STEPS can help users complete SQL tasks more efficiently and correctly.

We further investigated whether the SQL expertise of users has an impact on user performance. Fig. 6 shows the average number of tasks correctly completed by users with different levels of SQL

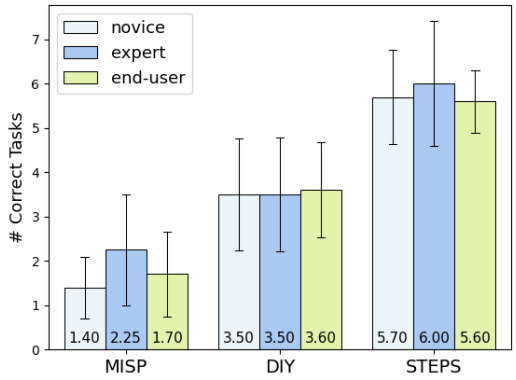

Figure 6: Tasks correctly completed by users with different levels of SQL expertise.

expertise as mentioned in Section 5. The results indicate that the performance does not depend on the expertise and different groups perform similarly.

Based on the survey responses, all participants ranked STEPS as the most usable and useful tool. As shown in Figure 5, participants felt the most confident and successful while experiencing the least mental load when using STEPS.

Appendix F shows the details of user study results.

6 Discussion and Limitations

Both the quantitative experiments and the user study demonstrate STEPS can significantly improve the accuracy of SQL generation to an unprecedented level. This is largely attributed to the interaction design, which allows users to precisely pinpoint which part of the SQL is wrong and only regenerates the incorrect clauses rather than the entire SQL query. While simple errors are prevalent in SQL generation, our ablation study (Table 3) shows that fixing simple errors alone is insufficient. Specifically, only using our direct transform approach, which only fixes simple errors, can only achieve 78.8% exact set accuracy. Yet using our hybrid method, which combines direct transformation with a neural approach to fix more complex errors, achieves 98.1%.

We believe this interactive design can be potentially applied to code generation tasks in other domains such as WebAPI (Su et al., 2017) and SPARQL (Ngonga Ngomo et al., 2013; Mo et al., 2022).

The major limitation of our work lies in the user simulation of the quantitative experiment. It assumes that users can provide perfect feedback—precisely locating an error in the SQL query and

clearly explaining the correct behavior. Yet in practice, users may miss or incorrectly specify an erroneous clause and provide ambiguous explanations. Our user study addresses this limitation by measuring actual user success. As expected, user accuracy from the user study (85.8%) is lower than the experiment with a simulated user (97.3%).

In this work, we only experiment on Spider (Yu et al., 2018) because it has become the de facto standard for measuring single-turn text-to-SQL performance. Since Spider is a complex and cross-domain dataset, we believe our work can be easily generalized to other simpler (Zhong et al., 2017) or domain-specific (Zelle and Mooney, 1996) datasets. Compared to our user study, users in real-world scenarios are often more familiar with their working context (e.g., database schema). Therefore, we expect that users will achieve better performance than in our user study.

In future work, it is worthwhile to develop mechanisms to account for potential errors and ambiguity in user feedback. A promising direction is to enable STEPS to ask clarification questions when the user feedback is ambiguous or incorrect. Furthermore, one may also consider augmenting our dataset with ambiguous feedback and re-training the text-to-clause model to improve its capability to interpret ambiguous feedback.

7 Conclusion

This work presents STEPS, a new interactive approach for text-to-SQL generation. STEPS decomposes the original text-to-SQL task into smaller text-to-clause tasks and enables users to validate and refine a generated query via editable explanations. An experiment on Spider and a user study show STEPS can significantly boost the accuracy of end-to-end models by incorporating user feedback. Furthermore, STEPS outperforms three state-of-the-art approaches for interactive SQL generation.

8 Ethical Consideration

The interactive text-to-SQL system proposed by this work poses minimal risks to human users and society. Instead, it will significantly lower the barrier of querying database systems and empower a great number of people, especially those without technical backgrounds, to access and analyze data. To evaluate the usability of our system, we conducted a human-subject study with real users. To minimize the risks to human subjects, we strictly followed the community standards with the approval from the IRB office in our institute. Specifically, in the recruitment email, we shared a consent form that includes detailed information about the study procedure, potential risks, data usage, and confidentiality. We obtained consent from each user before proceeding with the study. All collected data were anonymized and de-identified to protect the privacy of users.

References

Yitao Cai and Xiaojun Wan. 2020. Igsql: Database schema interaction graph based neural model for context-dependent text-to-sql generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6903–6912.

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*, pages 2065–2077, Online. Association for Computational Linguistics.

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*, pages 5599–5610, Online. Association for Computational Linguistics.

Jiaqi Guo, Ziliang Si, Yu Wang, Qian Liu, Ming Fan, Jian-Guang Lou, Zijiang Yang, and Ting Liu. 2021. Chase: A large-scale and pragmatic chinese dataset for cross-database context-dependent text-to-sql. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2316–2331.

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)*, pages 1339–1349, Melbourne, Australia. Association for Computational Linguistics.

Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. 1978. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2):105–147.

Binyuan Hui, Ruiying Geng, Qiyu Ren, Binhua Li, Yongbin Li, Jian Sun, Fei Huang, Luo Si, Pengfei

Zhu, and Xiaodan Zhu. 2021. Dynamic hybrid relation exploration network for cross-domain context-dependent semantic parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13116–13124.

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.

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)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.

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*, SIGMOD '12, page 673–676, New York, NY, USA. Association for Computing Machinery.

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)*, pages 333–344.

Fei Li and H. V. Jagadish. 2014a. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84.

Fei Li and Hosagrahar V Jagadish. 2014b. Nalir: An interactive natural language interface for querying relational databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, page 709–712, New York, NY, USA. Association for Computing Machinery.

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)*, pages 6913–6922, Online. Association for Computational Linguistics.

Lingbo Mo, Ashley Lewis, Huan Sun, and Michael White. 2022. Towards transparent interactive semantic parsing via step-by-step correction. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 322–342, Dublin, Ireland. Association for Computational Linguistics.

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*, IUI '21, page 597–607, New York, NY, USA. Association for Computing Machinery.

Saul B. Needleman and Christian D. Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453.

Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, Christina Unger, Jens Lehmann, and Daniel Gerber. 2013. Sparql2nl: Verbalizing sparql queries. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, page 329–332, New York, NY, USA. Association for Computing Machinery.

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*, pages 311–324, Online. Association for Computational Linguistics.

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*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Alkis Simitsis and Yannis Ioannidis. 2009. Dbmss should talk back too. In *10.48550/ARXIV.0909.1786*. arXiv.

Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, Michael Gamon, and Mark Encarnacion. 2017. Building natural language interfaces to web apis. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 177–186, New York, NY, USA. Association for Computing Machinery.

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*, pages 7567–7578, Online. Association for Computational Linguistics.

Run-Ze Wang, Zhen-Hua Ling, Jingbo Zhou, and Yu Hu. 2021a. Tracking interaction states for multi-turn text-to-sql semantic parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13979–13987.

Xiaxia Wang, Sai Wu, Lidan Shou, and Ke Chen. 2021b. An interactive nl2sql approach with reuse strategy. In *Database Systems for Advanced Applications: 26th International Conference, DASFAA 2021, Taipei, Taiwan, April 11–14, 2021, Proceedings, Part II*, page 280–288, Berlin, Heidelberg. Springer-Verlag.

William Woods, Ronald Kaplan, and Bonnie Webber. 1972. The lunar science natural language information system: Final report.

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)*, pages 5447–5458, Hong Kong, China. Association for Computational Linguistics.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. 2020. Grappa: Grammar-augmented pre-training for table semantic parsing. *CoRR*, abs/2009.13845.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019a. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979, Hong Kong, China. Association for Computational Linguistics.

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*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019b. SParC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy. Association for Computational Linguistics.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, page 1050–1055. AAAI Press.

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)*, pages 5338–5349.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. In *arxiv preprint, arxiv/1709.00103*. arXiv.

A SQL Grammar and Translation Rules

⟨ sql ⟩ := SELECT ⟨ nouns ⟩ ⟨ sub ⟩
| ⟨ sql ⟩ INTERSECT ⟨ sql ⟩
| ⟨ sql ⟩ UNION ⟨ sql ⟩
| ⟨ sql ⟩ EXCEPT ⟨ sql ⟩
⟨ sub ⟩ := ϵ
| FROM ⟨ noun ⟩ ⟨ sub ⟩
| WHERE ⟨ condition ⟩ ⟨ sub ⟩
| JOIN ⟨ noun ⟩ ON ⟨ condition ⟩ ⟨ sub ⟩
| GROUP BY ⟨ noun ⟩ ⟨ sub ⟩
| HAVING ⟨ condition ⟩ ⟨ sub ⟩
| ORDER BY ⟨ noun ⟩ ⟨ sorting ⟩ ⟨ sub ⟩
| LIMIT NUM
⟨ nouns ⟩ := DISTINCT ⟨ nouns ⟩
| ⟨ noun ⟩,⟨ nouns ⟩
| ⟨ noun ⟩
| ⟨ func ⟩ (⟨ noun ⟩)
⟨ condition ⟩ := ⟨ noun ⟩ ⟨ op ⟩ NUM
| ⟨ noun ⟩ ⟨ op ⟩ ⟨ noun ⟩
| ⟨ noun ⟩ ⟨ op ⟩ ⟨ sql ⟩
| BETWEEN ⟨ noun ⟩ AND ⟨ noun ⟩
| ⟨ condition ⟩ AND ⟨ condition ⟩
| ⟨ condition ⟩ OR ⟨ condition ⟩
| NOT ⟨ condition ⟩
⟨ sorting ⟩ := ASC | DESC | ϵ
⟨ func ⟩ := COUNT | AVG | MAX | MIN | SUM
⟨ op ⟩ := >= | <= | > | < | = | !=
⟨ noun ⟩ := STRING | STRING.STRING | *

Table 5: A Simplified SQL Grammar

Table 5 shows a simplified version of the SQL grammar. In this grammar, italicized text with angle brackets, such as $\langle sql \rangle$, represents non-terminals which can be further expanded based on derivation rules. Text without brackets, such as the SELECT keyword, represents terminals that cannot be further expanded. Using the derivation rules in Table 5, STEPS decomposes a SQL query into 6 types of SQL clauses: (1) FROM-JOIN-ON, (2) WHERE, (3) GROUP BY, (4) HAVING, (5) ORDER BY, (6) SELECT. We do not separate the JOIN clause from the FROM clause, since it is easier to translate them together in the later step. Furthermore, for nested

queries with INTERSECT, UNION, EXCEPT, NOT IN keywords, STEPS first decomposes them into subqueries and then decompose each subquery to the 6 types of clauses.

STEPS translates each SQL clause to NL explanation based on translation rules and templates in Table 6 and Table 7. Table 6 shows the translation rules for individual SQL tokens, e.g., keywords, operators, built-in functions, etc. Specifically, {col} and {T} mean translating a column or table name to a more readable name. We pre-defined mapping between each table and column in a database to a more readable name. Such a mapping can be easily defined based on the database schema and only needs to be defined once. If no such mapping is available, STEPS will reuse the same column/table name as defined in the database schema. Table 7 shows the translation templates for nested queries. The TRANSLATE function means recursively invoking the explanation generation method on the subquery.

B Synonym Substitution Rules for Paraphrasing

To increase the NL explanation diversity in our training dataset, we paraphrase each machine-generated explanation by randomly replacing the NL explanation template words with substitute synonyms listed in Table 8. For example, the machine-generated explanation "*return name*" can be paraphrased to "*find name*" by replacing "*return*" with "*find*".

C Experiment Setup & Hyperparameters

We run our experiment on a server with Ubuntu 20.04, 2 NVIDIA Tesla T4 GPUs (16 GB), Intel Core i7-11700K GPU, and 64 GB memory.

For the text-to-clause model, we follow the same architecture of SmBoP (Rubin and Berant, 2021). Specifically, our model consists of 24 transformer layers, followed by another 8 RAT-SQL (Wang et al., 2020) layers. Each transformer has 1 feed-forward layer, 8 attention heads, and 256 dimensions. For each user-given NL question, it is encoded together with the database schema using GRAPPA (Yu et al., 2020).

We finetuned the text-to-clause model and selected the best-performing model with the following hyperparameters: *optimizer* = *Adam*, *learning rate* = $1.8e-4$, *dropout rate* = 0.1, *beam size* = 26, *epoch* = 240, *batch size* = 12.

SQL Elements	Translation
SELECT	Return
FROM	In table
JOIN	and table
WHERE	Keep the records where
GROUP BY	Group the records based on
HAVING	Keep the groups where
ORDER BY	Sort the records based on
LIMIT 1	return the first record
LIMIT num	return the top num records
*	all the records
col_1, col_2	the {col_1} and the {col_2}
c_1 c_2 c_3	the {c_1}, the {c_2} and the {c_3}
T.col	{col} of {T}
COUNT(col)	the number of {col}
COUNT(*)	the number of records
AVG(col)	the average value of {col}
MAX(col)	the maximum value of {col}
MIN(col)	the minimum value of {col}
SUM(col)	the sum value of {col}
ASC	in ascending order
DESC	in descending order
=	is
!=	is not
>	is greater than
>=	is greater than or equal to
<	is less than
<=	is less than or equal to
IN	is in
NOT IN	is not in
BETWEEN	is between
LIKE	is in the form of
NOT LIKE	is not in the form of

Table 6: Translation rules for SQL elements

D The Impact of Paraphrasing on Model Performance

To investigate the impact of paraphrasing on model performance, we trained and tested the text-to-clause models under 3 conditions: (1) the explanation is generated by STEPS and not paraphrased, (2) the machine-generated explanation is paraphrased by the replacement rules in Table 8, and (3) the machine-generated explanation is paraphrased by QuillBot. Then we evaluate the exact set matching match accuracy of generated clauses in Table 1. Furthermore, we evaluate the end-to-end SQL generation accuracy in our user simulation experiment under 3 conditions. Overall, paraphrasing does not greatly impact the performance of text-to-clause SQL.

SQL compound	Translation
q_1 INTERSECT q_2	Start the first query: TRANSLATE(q_1); Start the second query; TRANSLATE(q_2); Return the intersection of them;
q_1 UNION q_2	Start the first query q_1: TRANSLATE(q_1); Start the second query: TRANSLATE(q_2); Return the union of them.
q_1 EXCEPT q_2	Start the first query: TRANSLATE(q_1); Start the second query: TRANSLATE(q_2); Return the records in q_1 but not in q_2.
... col IN/NOT IN q_1	Start the first query: TRANSLATE(q_1); Start the second query: TRANSLATE(...); Keep the records where {col} in/not in q_1.

Table 7: NL explanation translation rules for SQL compound

E User Interfaces of STEPS and Baselines

This section demonstrates the user interface (UI) of STEPS, DIY, and MISP used in our user study.

As shown in Fig. 7, the UI of STEPS has 4 views. First, the upper left view allows users to select a database and inspect the data records in each table. Users are allowed to search, rank, and filter data records in the table. This view helps users explore the database and manually validate the query result based on the original data. Second, the upper right view provides a dialog for users to ask questions in natural language. For each question, STEPS automatically generates a SQL query. Third, the lower left view shows the query result of a generated SQL. Users can inspect the query result to validate whether the generated query is correct or not. Fourth, the lower right view renders the core functionality of STEPS, which provides an editable step-by-step explanation for the generated SQL query. Users can easily read the explanation and identify whether there is any erroneous or missing step in the query. If users find an error in a step, they can directly edit the explanation of that step. Users can also add or remove a step. If users click the ADD button next to a step, an empty text field will appear right below this step and the user can write the description for this new step. If users click the REMOVE button next to a step, this step will be removed. Furthermore, users can check the intermediate query result of a step by clicking the circled step number icon. For example, if users click the green number ①, STEPS just returns all the data in the AIRPORT table. Additionally, users can undo and redo previous edits using the stepper below.

As shown in Fig. 8, MISP shares a similar UI as STEPS. MISP also allows users to select a database, inspect data in a table, and view the query result. The main difference from STEPS 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, it will proactively predict which part of the SQL is wrong and ask users to select alternative generations to fix the error. Furthermore, MISP does not provide an NL explanation of the generated SQL. Users have to read and inspect the generated SQL, which is difficult for end-users who do not understand the syntax and semantics of SQL.

Fig. 9 shows the UI of DIY. To reduce the information overload of inspecting a large database, DIY only samples a small amount of data from a user-selected 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 NL question with tokens in the generated SQL. Each matched NL tokens is augmented with a dropdown menu with alternative SQL tokens predicted by the base model. If the prediction of a token is wrong, users can click on the dropdown 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.

F Analysis of Post-study Survey Responses

We analyzed the post-task survey responses and interview recordings to understand why participants performed much better when using STEPS compared with using MISP and DIY. Specifically, 17 participants strongly agree that seeing the natural language explanation helps them understand the SQL query. Moreover, 22 participants explicitly wrote that they highly appreciated the step-by-step explanations provided by STEPS, since these explanations made SQL queries more understandable, editable, and learnable. P12 wrote, "*I liked that it shows the steps in human language so if there is a mistake I can edit it easily. Also, it was nice to see the generated SQL code I believe I could learn SQL using this tool also.*" By contrast, 14 of 24 par-

Template word	Substitute synonyms
return	get, find, find out, discover, show, show me, determine, demonstrate, give me, obtain, select, choose, search, choose, search, display, list, acquire, gain
keep the records where	make, make sure, where, filter the records where
greater than	more than, exceed, no less than, over, above, larger than, beyond, in excess of, transcend, surpass
less than	lower than, no more than, below, lesser, under, underneath, not so much as, beneath
ascending	increasing, ascendant, growing, rising, soaring, climbing, mounting
descending	decreasing, descendant, falling, declining, dropping, lessening, diminishing
maximum	max, maximum, utmost, greatest, most, topmost, highest, top, largest, biggest
minimum	lowest, smallest, least, min, minimal, bottom, bottommost, lowermost
number of	amount of, quantity of, total of
in the form of	appearing as, with the appearance of, in the shape of
that has	associated with, connected to
based on	according to, in terms of, specified by, built on, established on, considering, regarding
distinct	different, disparate, distinctive, particular, diverse, dissimilar, unique
all	each, every, any, whole, entire, total
group	batch, organize, categorize, classify, arrange, separate, label, tag, mark, pack, collect, assemble, distribute, gather, merge, put together, index, concentrate, combine
Sort	order, rank, sequence

Table 8: Replacement rules for paraphrasing NL explanation

	Acc$_{\text{set}}$
No paraphrasing	0.922
Paraphrasing with synonym substitution	0.915
Paraphrasing with QuillBot	0.906

Table 9: The exact set matching accuracy of the text-to-clause model when trained with three different datasets.

	Acc$_{set}$	Acc$_{exec}$
SmBoP+STEPS^{unpara}	0.981	0.973
SmBoP+STEPS^{substitute}	0.975	0.973
SmBoP+STEPS^{quillbot}	0.975	0.971

Table 10: The end-to-end SQL generation accuracy of STEPS when using the text-to-clause model trained on different datasets.

ticipants reported it was hard to understand and validate the generated SQL queries when using MISP or DIY. P1 wrote, "*Sometimes it generates very complex SQL that is difficult to read and check.*" P9 wrote, "*Sometimes it gives the wrong answer. As I'm no expert in SQL, I couldn't tell instantly if the queries were wrong, so I had to go back to the data and check manually.*" 12 participants reported that the feedback elicitation mechanism in MISP was not very efficient in solving SQL tasks. P16 wrote, "*I have to keep answering yes or no questions when using MISP.*"

11 of them reported the drop-down menu of DIY provides the limited capability to make changes. P3 said, "*[It is] hard to know how to make changes / resolve issues that were not covered by the drop-down menus.*"

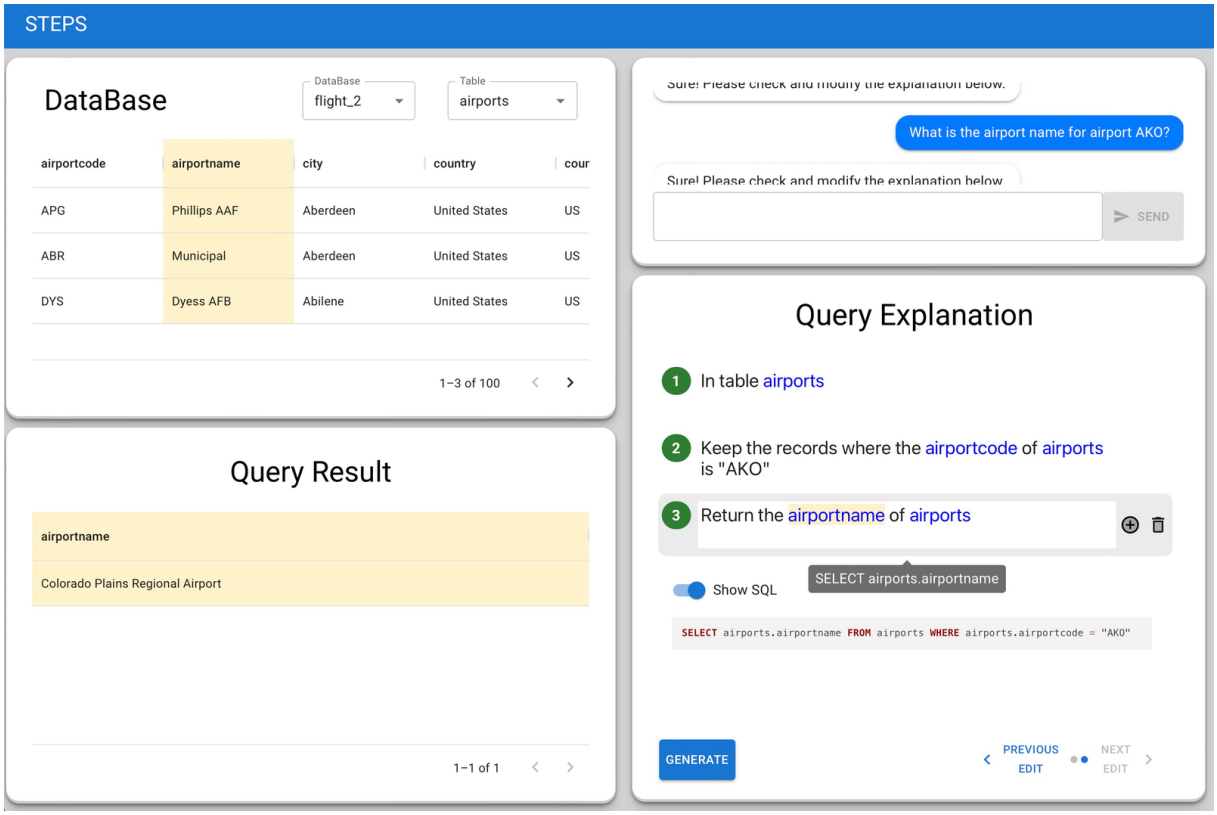

Figure 7: The UI of STEPS

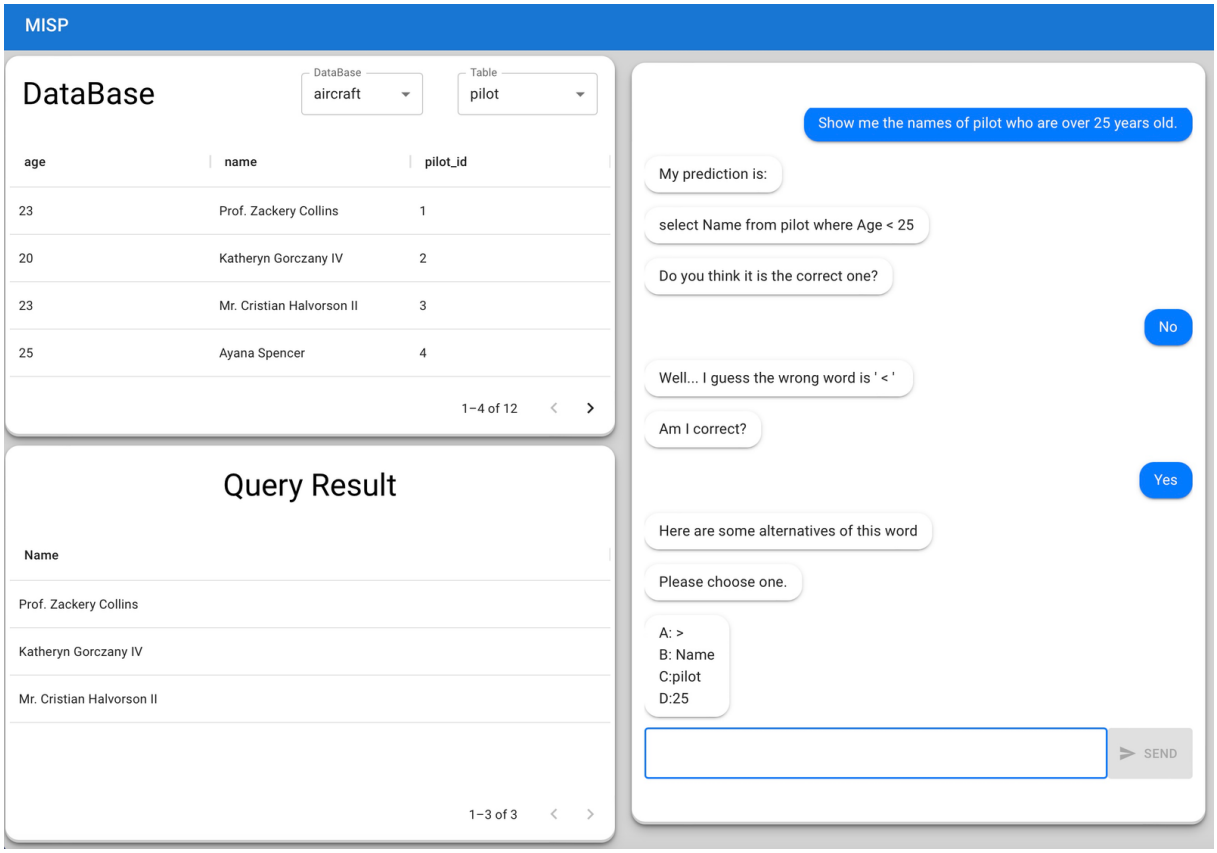

Figure 8: The UI of MISP

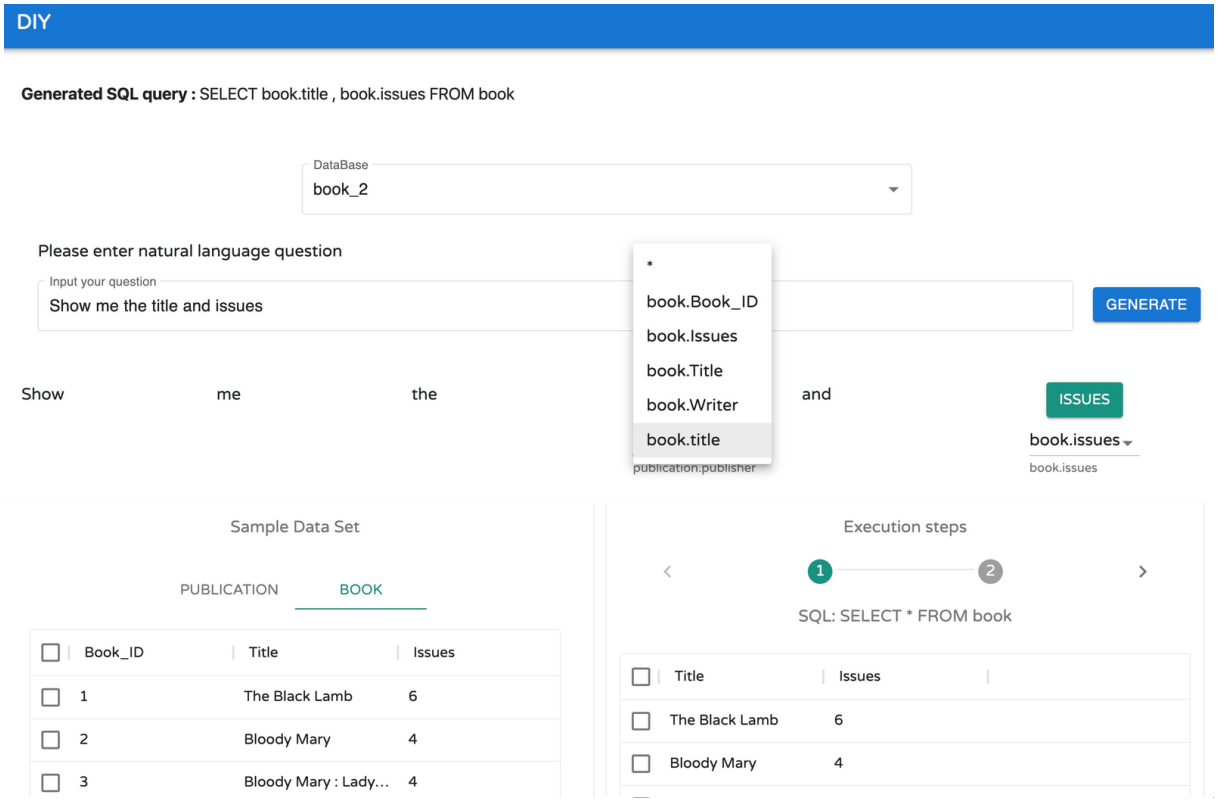

Figure 9: The UI of DIY