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 business, science, and more. However, many users cannot fully unleash the analytical power of relational databases, because 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 incorrect queries. To address these issues, we introduce a new interaction mechanism that allows users to directly edit a step-by-step explanation of a query to fix errors. Our experiments on multiple datasets, as well as a user study with 24 participants, demonstrate that our approach can achieve better performance than multiple SOTA approaches. Our code and datasets are available at https://github.com/magic-YuanTian/STEPS.

1 Introduction

Natural language interfaces significantly lower the barrier to accessing databases and performing data analytics tasks for users who are not familiar with database query languages. Many approaches have been proposed for generating SQL queries from natural language (Popescu et al., 2004; Giordani and Moschitti, 2012; Rubin and Berant, 2021; Scholak et al., 2021; Zhao et al., 2022). Using recent large language models, systems have reached 86.6% execution accuracy (Gao et al., 2023) on the Spider benchmark (Yu et al., 2018).

However, the rate of improvement has slowed, with a gain of only 10% since mid-2021. This is partly due to the inherent ambiguity of natural language and the complex structure of SQL queries (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).

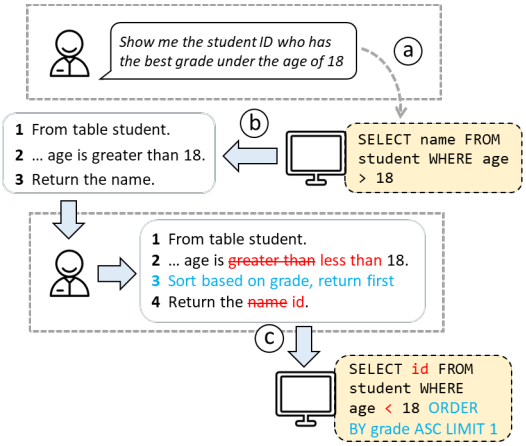

Figure 1: Refining a SQL query by directly editing a step-by-step explanation.

There has been growing interest in developing "human-in-the-loop" approaches that elicit user feedback to guide SQL generation. However, most approaches only support feedback in constrained forms, e.g., answering multiple-choice questions (MISP, PIIA, DialSQL Yao et al., 2019; Li et al., 2020; Gur et al., 2018), changing SQL elements in a drop-down menu (DIY, Narechania et al., 2021), etc. 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 as new utterances. However, since the feedback is open-ended, interpreting it can be just as hard as processing the original request.

In this paper, we seek to strike a balance between constrained feedback and open-ended feedback by proposing a new interaction mechanism: 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) a 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

Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pages 16149–16166
December 6-10, 2023

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 rule-based method to generate step-by-step explanations and uses a hybrid rule/neural 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 match accuracy, outperforming prior interactive text-to-SQL systems—MISP, DIY, and NL-EDIT—by 33.5%, 33.2%, and 31.3% respectively. We further evaluate STEPS on other datasets, including Spider-DK (Gan et al., 2021b), Spider-Syn (Gan et al., 2021a), and WikiSQL (Zhong et al., 2017). STEPS consistently achieves at least 96% exact set match accuracy and execution accuracy across all datasets.

Finally, we conducted a within-subjects 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 respectively,[1] with significantly higher self-reported confidence and lower mental load.

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 and a new training method for SQL correction; (3) we conduct a comprehensive evaluation with both simulated and real users and demonstrate its effectiveness over state-of-the-art interactive systems. Our dataset and code are publicly available.

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).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., 2020a; 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., 2021; Hui et al., 2021; Zhang et al., 2019; Cai and Wan, 2020; Wang et al., 2020b). 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) proposed to allow users to flag incorrect queries and continuously retrain the model. Both DIY (Narechania et al., 2021) and NaLIR (Li and Jagadish, 2014a,b) enable users to select alternative values or subexpressions to fix an incorrect SQL query. PIIA (Li et al., 2020), MISP (Yao et al., 2019), and DialSQL (Gur et al., 2018) proactively ask for user feedback via multiple-choice questions. A common limitation of these methods is that they only solicit feedback in constrained forms, hindering their flexibility and effectiveness in addressing the variability of SQL errors. In contrast, our approach allows more flexible feedback through direct edits to the explanations generated by the model.

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 utterances. Given an incorrect query, NL-EDIT allows users to provide a clarification utterance. Based on the utterance, the model generates a sequence of edits to the SQL query. Incorporating feedback expressed in a completely free-text utterance is challenging for two reasons:

[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 for comparisons.

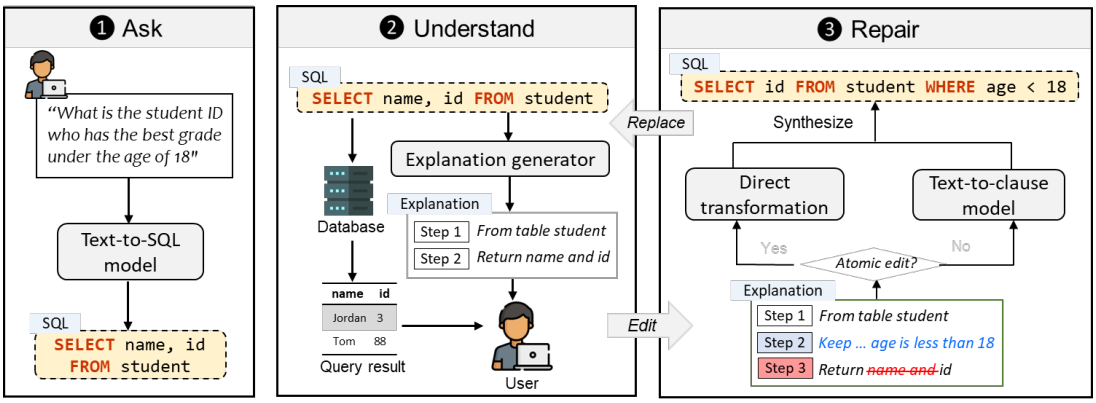

Figure 2: An Overview of Interactive SQL Generation and Refinement with Editable Step-by-Step Explanations

(1) the model needs to infer which part of the SQL query to fix; (2) the model needs to determine what changes are being requested. In contrast, STEPS 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 recent work that decomposes a user question to sub-questions on SPARQL (Mo et al., 2022). Their approach requires a crowd-sourced dataset to train a question decomposition model. In contrast, our rule-based method generates step-by-step explanations without the need for training a model. This also allows our system to map each entity in the explanation to the corresponding SQL element, making it easier for SQL correction (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 human language so that users can verify results. Kokkalis et al. (2012) and Koutrika et al. (2010) used a graph-based SQL translation approach, where each query 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 approaches have limited capability to handle arbitrary SQL queries. To address this limitation, we propose a rule-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. Another key difference is that none of the existing approaches supports *editable* explanations for SQL correction, which is a key feature to solicit user feedback in 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 update the SQL query, rather than regenerate the entire query from scratch. If multiple steps are incorrect, the user can add, remove, and modify all steps as needed.

3.1 Rule-based SQL Explanation

To generate explanations for arbitrarily complex SQL queries (e.g., a query with nested subqueries), we design a rule-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 6. 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 correspond-

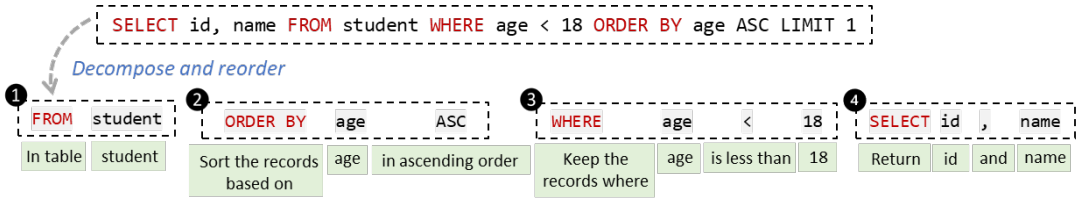

Figure 3: An example of the explanation generation process

ing NL description based on a set of translation rules (see Table 7 in the appendices). For example, SELECT is translated to "Return", and Order By is translated to "Sort the records based on." 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 to reflect their execution order. We believe 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

Users make edits to the explanation produced by our system to make it consistent with their goal. Given these edits, STEPS uses a hybrid method to generate the corresponding SQL clause. For simple edits, such as replacing a column name, STEPS directly edits the original clause to fix the error using three direct transformation rules (§ 3.2.1). For more complex edits, STEPS uses a neural text-to-clause model to generate the clause based on the user-corrected explanation (§ 3.2.2).

The hybrid method is inspired by the findings from our recent study (Ning et al., 2023). Specifically, a large portion of SQL generation errors are simple errors (e.g., incorrect column names and operators), which can be fixed with small edits. After SQL decomposition by our approach, many larger errors are further decomposed into a set of simpler errors, contained within separate clauses. 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 4).

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

Algorithm 1: Direct transformation

Input: The original explanation e_o;
The new edited explanation e_n;
The original SQL clause s;
Output: the updated SQL clause

```
1  C_o ← CHUNK(e_o)
2  C_n ← CHUNK(e_n)
3  foreach (c_o, c_n) in ALIGN(C_o, C_n) do
4      // Replace ;
5      if BOTHCOLUMN(c_o, c_n) or
6      BOTHTABLE(c_o, c_n) or
7      BOTHLITERAL(c_o, c_n) then
8          s ← s.REPLACE(c_o, c_n) ;
9      // Add ;
10     else if c_o is ∅ and ISCOLUMN(c_n) then
11         if s.STARTWITH("Select") then
12             s ← s.APPEND(c_n)
13     // Remove ;
14     else if c_n is ∅ and ISCOLUMN(c_o) then
15         s ← s.REMOVE(c_o) ;
16 end
17 return
```

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.

Algorithm 1 describes our direct transformation algorithm. After chunking the text (Lines 1-2), STEPS aligns and compares the chunks in the original explanation with those in the user-corrected explanation, using the Needleman and Wunsch (1970) algorithm (Line 3). This allows STEPS to detect any replacements (Line 4), additions (Line 9), or removals (Line 13) of database entities in the explanation. Based on this information, STEPS automatically edits the corresponding SQL clause without calling a neural model (Lines 8, 12, 15). More details of this algorithm can be found in Appendix E.

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

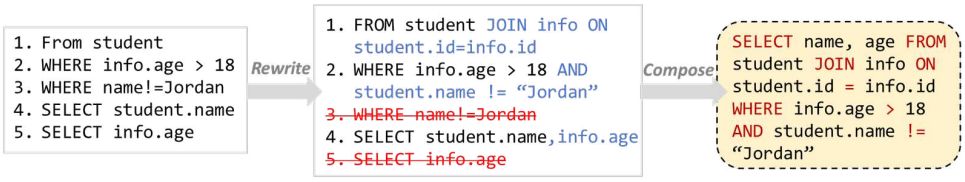

Figure 4: An example of SQL clause rewriting and composition

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 model, we automatically created a dataset with 83K text-clause pairs based on Spider (Yu et al., 2018). For each SQL query in Spider, we use the explanation generation method in Section 3.1 to decompose the query 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 use a rule-based method to replace words with their synonyms (details in Table 9 in the appendices). Second, we paraphrase the explanation using an automatic paraphrasing tool: QuillBot[3]. We train the text-to-clause model using Adam with a learning rate of $1.8e-4$ and a dropout rate of 0.1. We perform 10-fold cross-validation and the exact set match accuracy of our text-to-clause model is 90.6% (see Appendix D for details).

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 avoiding syntactic errors using three 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 three 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 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). We also explored the impact on STEPS of different text-to-SQL models, different task difficulties, and four different benchmarks (Yu et al., 2018; Gan et al., 2021b,a; Zhong et al., 2017). Finally, we conducted an ablation study for our hybrid method.

4.1 Automated User Simulation & Setup

For our quantitative evaluation of STEPS, we developed an automated script to simulate user feedback following the user simulation method of Yao et al. (2019). This setup assumes we have a user who perfectly identifies all errors and provides clear corrections. The purpose of this experiment is to measure the upper bound of system performance without human errors.

We do this as follows. 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 a clause in the generated query is not semantically equivalent to the corresponding clause in the ground truth (i.e., there is an error).

We simulated three types of mismatches. 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 is missing a clause from the ground truth, our script will generate the 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 a clause that is inconsistent with the ground truth, our script will generate the 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 Comparison Systems

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

Among tools that allow users to give feedback by answering multiple-choice questions (Gur et al., 2018; Li et al., 2020; Yao et al., 2019), we select MISP (Yao et al., 2019) to compare with because it has the best performance in simulation. During the interaction, MISP asks users to clarify whether a column should be considered in the query, and the user can answer yes or no. The user's answer is used to constrain the decoding process by adjusting the probability of code tokens induced by the answer. We used the original implementation of MISP from their GitHub repository. Furthermore, since their GitHub repository provides a user simulation script, we reuse it for the user simulation in our experiments.

DIY (Narechania et al., 2021) enables users to refine a generated SQL query by showing the table names, column names, operators, and aggregate functions that correspond to words in the NL question and allowing the user to select alternatives from drop-down menus. We reimplemented DIY since no open-source implementation is available. We cannot directly compare with the user performance from the DIY paper because they did not report any objective measures, such as task completion rates and time (Narechania et al., 2021). To construct the word-entity mapping in DIY, 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 giving feedback in natural language. User feedback is parsed into a set of simple edits (e.g., add, remove) that are applied to the SQL query. We worked with the NL-EDIT authors to run their system, but were unable to resolve issues due to missing code and other run-time errors. We report results for NL-EDIT using the accuracy numbers from the NL-EDIT paper.

	Acc_{set}
EditSQL (Zhang et al., 2019)	0.576
Human-in-the-Loop Methods	
+ MISP (Yao et al., 2019)	0.644
+ DIY (Narechania et al., 2021)	0.647
+ NL-EDIT (Elgohary et al., 2021)	0.666
+ STEPS (Ours)	**0.979**
AI-Only Methods	
Graphix-3B + PICARD (Li et al., 2023b)	0.771
SHiP + PICARD (Zhao et al., 2022)	0.772
RESDSQL-3B + NatSQL (Li et al., 2023a)	0.805

Table 1: Exact Set Matching Accuracy Comparison. Note, these results are on the dev set as we are unable to use the hidden test set in the human experiments.

4.3 Results

Comparison with the Three SOTA Interactive Approaches. Table 1 shows the exact set match accuracy of STEPS, MISP, DIY, and NL-EDIT. 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 previous approaches by at least 31%.

Comparison with Strong Text-to-SQL Models. Table 1 also shows the exact set match accuracy of three high-performing text-to-SQL models (Li et al., 2023b; Zhao et al., 2022; Li et al., 2023a).[4] Compared with these models, STEPS achieved 17%-20% accuracy improvement by soliciting user feedback. This indicates that allowing users to edit step-by-step explanations can produce results that are far better than the best pure-AI models while also providing users with confidence that the query is doing what they want.

Evaluation with Different Base Models & Task Difficulty Levels. 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

[4] As the test set of Spider is not released, we selected the top three models based on their exact set match accuracy on the development set at the time of our experiments.

	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}	
	SmBoP	+ STEPS	SmBoP	+ STEPS
WikiSQL	0.862	0.983	0.895	0.980
Spider	0.745	0.981	0.657	0.973
Spider-DK	0.534	0.987	0.537	0.976
Spider-Syn	0.572	0.969	0.600	0.972

Table 3: Evaluation on different datasets

	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 4: Ablation Study of the Hybrid Method

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 various 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. For hard and extra hard tasks, the generated SQL queries often include more errors. It can be challenging for other approaches to fix all of them at once. In our case, decomposing the original task into smaller steps makes fixing multiple errors as easy as fixing one.

Generalizability to Different Datasets. To further demonstrate the generalizability of STEPS, we evaluate STEPS on three other datasets——Spider-DK (Gan et al., 2021b), Spider-Syn (Gan et al., 2021a), and WikiSQL (Zhong et al., 2017). Note that since STEPS is trained on Spider, its models are out-of-domain when applying to different datasets. Table 3 demonstrates that STEPSachieves comparable performance across datasets.

Ablation Study for the Hybrid Method. Table 4 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 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 increases efficiency without decreasing accuracy.

5 User Study

To evaluate the usability and accuracy of STEPS when interacting with real users, we conducted a within-subjects user study with 24 participants.[6]

5.1 Participants

We recruited 24 participants (22M, 2F) through mailing lists at Purdue University. In the recruitment email, we shared a consent form that included 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. Each participant received a $25 gift card as compensation for their time.

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 syntactic details when writing a SQL query (novice); 4 could fluently write SQL queries (expert).

5.2 Comparison Systems

We used MISP (Yao et al., 2019) and DIY (Narechania et al., 2021) as comparison systems. As explained in Section 4.2, we did not use NL-EDIT, since we were unable to reproduce it. To ensure a

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

[6] Our study was approved by Purdue University's IRB.

fair comparison, we developed user interfaces with the same visual style for STEPS, MISP (Yao et al., 2019), and DIY (Narechania et al., 2021). User interface screenshots are provided in Appendix G.

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. The time limit was decided by 4 pilot studies, allowing sufficient time to complete multiple 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 in 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 started with participants watching a tutorial video of the assigned tool (6 min for STEPS, 3 min for MISP, and 2 min for DIY). The STEPS video was longer simply because STEPS had more features. During all tutorials, we allowed users to pause the video and ask questions. Participants were then 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 and results, participants could validate and repair the generated query using the tool. Participants were allowed to skip a task if they found it too hard to solve. Participants could view the produced SQL, but were not given the ability to edit the SQL directly.

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.

5.4 Results

Task Completion Rate Analysis. Table 5 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 provided sufficient support for users to tackle challenging tasks so that users did not give up quickly. The ANOVA test indicates that the mean differences in Table 5 are statistically significant among all three conditions (p-value < 0.01). These results indicate that STEPS can help users complete SQL tasks more efficiently and correctly than prior methods.

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

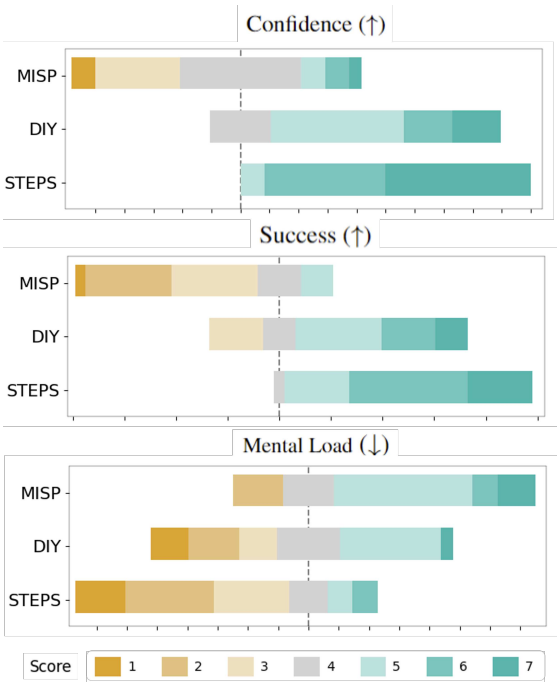

Figure 5: User Perception.

The Impact of SQL Expertise. We further investigated whether the SQL expertise of users has an impact on user performance. We found that all three user groups performed similarly in each condition. This implies that SQL expertise does not have a significant impact on user performance

when interacting with STEPS. For a visualization of the results, see Fig. 6 in the appendices.

User Perception. In the post-study survey, 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.

6 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. 17 participants strongly agreed that seeing the natural language explanation helped them understand the SQL query, and 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.*" In contrast, 14 of 24 participants 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. P16 wrote, "*I have to keep answering yes or no questions when using MISP.*" 11 of them reported the drop-down menus provided by DIY limited their ability 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.*"

7 Discussion

Both the quantitative experiments and the user study demonstrate STEPS can significantly improve the accuracy of SQL generation. 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. In contrast, existing approaches do not support expressive ease or error isolation. Users either cannot regenerate new content (e.g., DIY), or can only regenerate the entire query rather than just the erroneous part (e.g., MISP). Ning et al. (2023) showed that this lack of error isolation often introduces new errors, which frustrates users and makes errors hard to fix.

Error Analysis. While simple errors are prevalent in SQL generation, our ablation study (Table 4) shows that only fixing simple errors is insufficient, which motivates the design of our hybrid method. Our hybrid method can handle a broad range of errors because users can flexibly correct entities or clauses in a query. This ability helps reduce the difficulty of tasks by dividing complex errors into simpler ones, allowing users to solve them separately.

In our automated user simulation, STEPS failed in a few cases when the text-to-clause model predicted the wrong clause type. For example, the paraphrased ground truth explanation of one step was: "*Ensure that all categories where the total cost of therapy exceeds 1000 are included.*" The text-to-clause model predicted a `WHERE` clause instead of a `HAVING` clause.

In the user study, one common challenge arose when multiple tables in the database had the same column name. If users did not look carefully at the database schema, they may have not explicitly indicated the table to be used. That creates an ambiguity for the model.

Other Datasets and Domains. Our system should work for any SQL dataset, as our approach is domain-agnostic and covers general SQL structures. For other forms of code, such as WebAPI (Su et al., 2017) and SPARQL (Ngonga Ngomo et al., 2013; Mo et al., 2022), the general idea is applicable, but new models would be needed for (a) code generation, (b) explanation generation, and (c) code correction.

8 Conclusion

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

9 Limitations

Our automated user simulation is an optimistic experiment that does not account for user errors, such as not being able to identify mistakes in the explanation. The simulation was designed to test a scenario in which a user can perfectly identify which step of the explanation is wrong and accurately describe a corrected version in natural language. Creating such a perfect user required the use of the ground truth, both for the identification step and to generate the natural language correction. This simulation is not representative of real-world use. That limitaton was the motivation for our study with real users, in which we had actual people use different tools without information about correct answers. As shown in Table 5, the accuracy of the user study is lower than the simulation, but STEPS is still very effective and outperforms other tools. We choose to include the simulation study because it shows the potential for STEPS to make corrections if there is no human error.

In this paper, we only evaluate STEPS on single-turn SQL generation. In future work, our approach can be extended to multi-turn SQL generation by incorporating contextual information when editing the natural language explanation.

While our approach is designed to be general for SQL generation and potentially other code generation tasks, the current version only supports SQL keywords that appear in the Spider dataset. Like other text-to-SQL datasets, Spider only covers query operations (e.g., `SELECT`) and does not cover update operations (e.g., `INSERT`) for evaluation convenience. But it would be straightforward to cover unsupported operations by adding new translation rules.

10 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 Purdue University IRB office. 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.

Acknowledgments

This material is based in part on work supported by an Amazon Research Award, the Australian Research Council through a Discovery Early Career Researcher Award and by the Defense Advanced Research Projects Agency (grant #HR00112290056).

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, Online. Association for Computational Linguistics.

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.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-SQL models against synonym substitution. 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 2505–2515, Online. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-SQL generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8926–8931, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv*.

Alessandra Giordani and Alessandro Moschitti. 2012. Translating questions to SQL queries with generative parsers discriminatively reranked. In *Proceedings of COLING 2012: Posters*, pages 401–410, Mumbai, India. The COLING 2012 Organizing Committee.

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, Online. Association for Computational Linguistics.

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 network for cross-domain context-dependent semantic parsing. *CoRR*, abs/2101.01686.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. In *KR2ML Workshop at NeurIPS*.

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.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. RESDSQL: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press.

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.

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, page 633–649, New York, NY, USA. Association for Computing Machinery.

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*, pages 141–147, Geneva, Switzerland. COLING.

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. 2020a. 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.

Runze Wang, Zhen-Hua Ling, Jingbo Zhou, and Yu Hu. 2020b. Tracking interaction states for multi-turn text-to-sql semantic parsing. *CoRR*, abs/2012.04995.

Xiaxia Wang, Sai Wu, Lidan Shou, and Ke Chen. 2021. 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.

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.

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, Hong Kong, China. Association for Computational Linguistics.

Yiyun Zhao, Jiarong Jiang, Yiqun Hu, Wuwei Lan, Henry Zhu, Anuj Chauhan, Alexander Li, Lin Pan, Jun Wang, Chung-Wei Hang, Sheng Zhang, Marvin Dong, Joe Lilien, Patrick Ng, Zhiguo Wang, Vittorio Castelli, and Bing Xiang. 2022. Importance of synthesizing high-quality data for text-to-sql parsing.

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 6: A Simplified SQL Grammar

Table 6 shows a simplified version of the SQL grammar. In this grammar, italicized text with angle brackets, such as ⟨*sql*⟩, 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 6, 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. 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 above.

STEPS translates each SQL clause to a natural language explanation based on translation rules and templates in Table 7 and Table 8. Table 7 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 8 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 9. 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., 2020a) 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 follow-

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 7: Translation rules for SQL elements

ing hyperparameters: *optimizer* = *Adam*, *learning rate* = $1.8e-4$, *dropout rate* = 0.1, *beam size* = 26, *epoch* = 240, *batch size* = 12.

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 9, and (3) the machine-generated explanation is paraphrased by QuillBot. Then we evaluate the exact set matching match accuracy of generated clauses in Table 10. Furthermore, we evaluate the end-to-end SQL generation accuracy in our user simulation experiment under 3 conditions, as shown in Table 11. 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 8: NL explanation translation rules for SQL compound

E Direct Transformation Algorithm

As mentioned in Sec. 3.2.1, we define three types of *atomic edits*. While one can always design new transformation rules to support other simple edits, here we focus on these three basic edit types in order to demonstrate the benefits of direct transformation.

Algorithm 1 describes the direct transformation algorithm. First, STEPS performs chunking on the original explanation e_o and the user-corrected explanation e_n (Line 1-2). We choose to split an explanation into phrases rather than individual words in order 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 (Line 5-8). 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 (Line 9-12). 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 (Line 13-15).

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 9: Replacement rules for paraphrasing NL explanation

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

Table 10: 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 11: The end-to-end SQL generation accuracy of STEPS when using the text-to-clause model trained on different datasets.

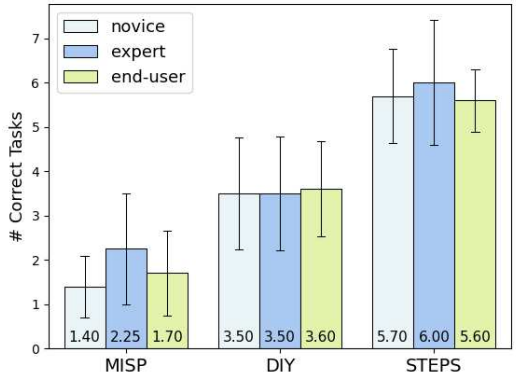

Figure 6: Tasks correctly completed by users with different levels of SQL expertise.

F Impact of SQL Expertise

Figure 6 shows a performance breakdown in our user study based on participants' SQL expertise. We observe that expertise does not impact performance, with consistent performance on all tools by all groups. The means for MISP do differ, but the distributions of scores overlap substantially.

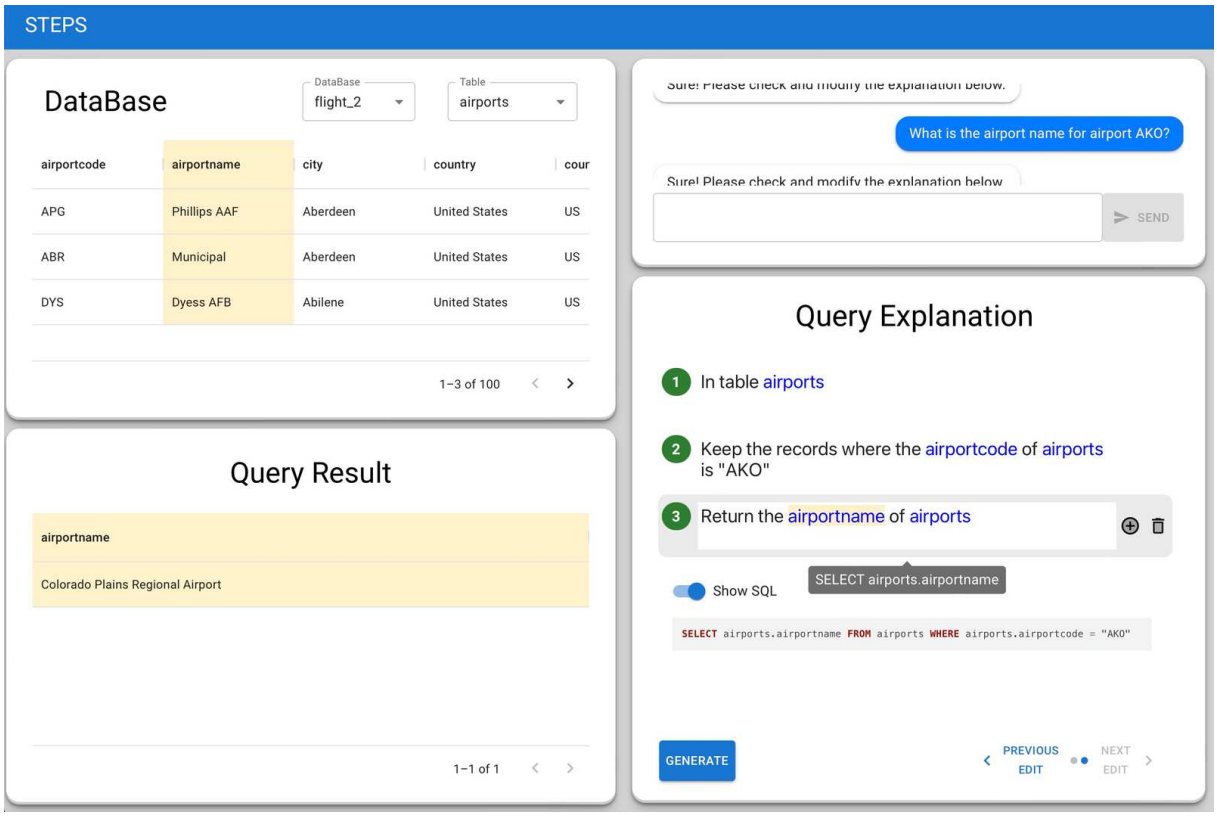

Figure 7: The UI of STEPS

G User Interfaces of STEPS and Baselines

This section describes 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.

Upper Left View This 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.

Upper Right View This provides a dialog box for users to ask questions in natural language. For each question, STEPS automatically generates a SQL query.

Lower Left View This shows the results of running a generated SQL query. Users can inspect the query result to validate whether the generated query is correct or not.

Lower Right View This renders the core functionality of STEPS: an editable step-by-step explanation for the generated SQL query. Users can easily read the explanation and identify whether there are any errors or missing steps 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 via the UI without needing to manually assign a step number. If a user clicks the ADD button next to a step, an empty text field will appear right below the step and the user can write the description for the new step. If a user clicks the REMOVE button next to a step, the step will be removed. We expect users to edit the steps in the correct location for reading clarity, but STEPS can also help rectify any errors or misoperations using the heuristics mentioned in Sec. 3.3. As shown in Fig. 4, if the user adds a new explanation "*return age*" that is parsed into "`SELECT info.age`", STEPS will automatically merge it with the existing `SELECT` clause and complete the `FROM` clause.

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, the MISP UI is very similar to STEPS. MISP also allows users to select a database, inspect data in a table, and view the query result. The main difference 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 predict which part of the SQL is wrong and ask users to select alternative generations to fix the error. 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 an SQL generation model. DIY automatically matches tokens in the NL question with tokens in the generated SQL. Each matched NL token 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.

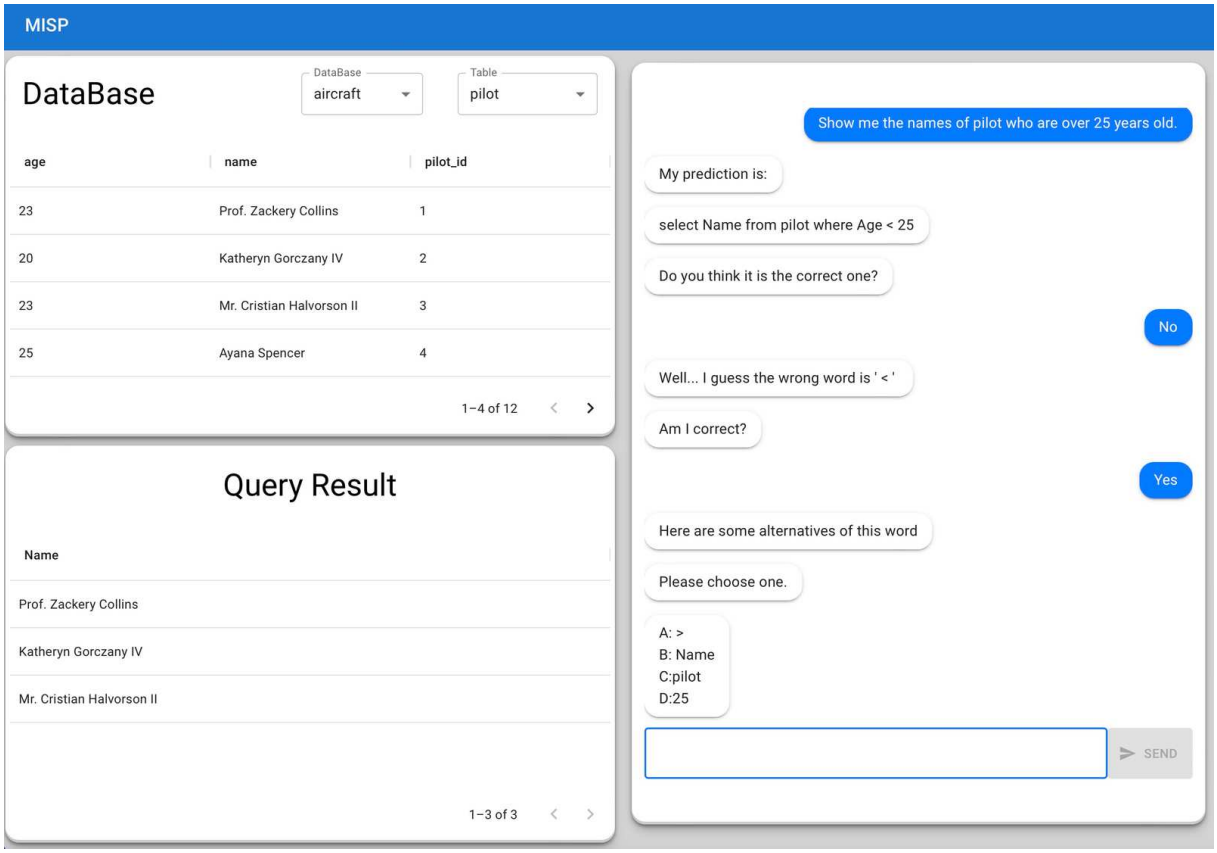

Figure 8: The UI of MISP

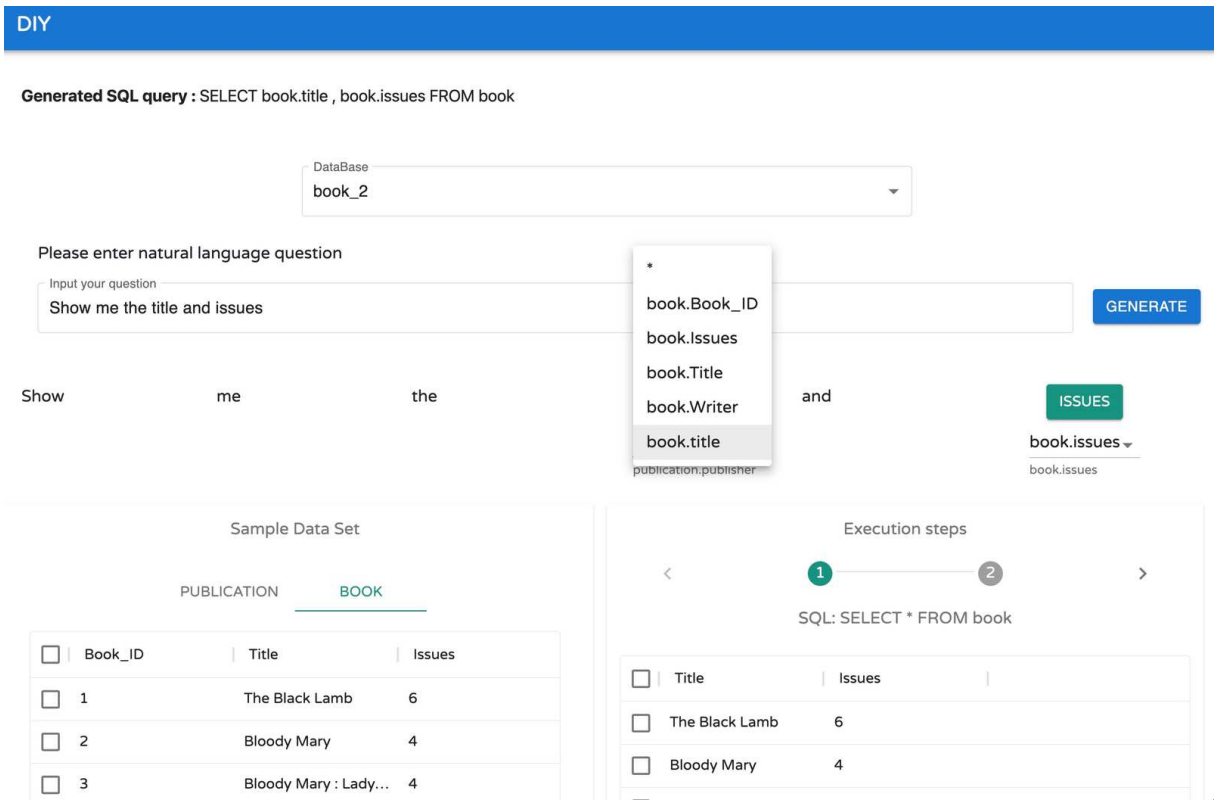

Figure 9: The UI of DIY