Sphinteract: Resolving Ambiguities in NL2SQL Through User Interaction

Fuheng Zhao
UC Santa Barbara
fuheng_zhao@ucsb.edu

Shaleen Deep
Microsoft
shaleen.deep@microsoft.com

Fotis Psallidas
Microsoft
fotis.psallidas@microsoft.com

Avrilia Floratou
Microsoft
avrilia.floratou@microsoft.com

Divyakant Agrawal
UC Santa Barbara
agrawal@cs.ucsb.edu

Amr El Abbadi
UC Santa Barbara
amr@cs.ucsb.edu

ABSTRACT

Translating natural language questions into SQL queries (NL2SQL) is a challenging task of great practical importance. Prior work has extensively studied how to address NL2SQL using Large Language Models (LLMs) with solutions ranging from careful prompt engineering, to fine-tuning existing LLMs, or even training custom models. However, a remaining challenging problem in NL2SQL is the inherent ambiguity in the natural language questions asked by users. In this paper, we introduce Sphinteract, a framework designed to assist LLMs in generating high-quality SQL answers that accurately reflect the user intent. Our key insight to resolve ambiguity is to take into account minimal user feedback interactively. We introduce the *Summarize, Review, Ask* (SRA) paradigm, which guides LLMs in identifying ambiguities in NL2SQL tasks and generates targeted questions for the user to answer. We propose three different methods of how to process user feedback and generate SQL queries based on user input. Our experiments on the challenging KaggleDBQA and BIRD benchmarks demonstrate that by means of asking clarification questions to the user, LLMs can efficiently incorporate the feedback, resulting in accuracy improvements of up to 42%.

PVLDB Reference Format:
Fuheng Zhao, Shaleen Deep, Fotis Psallidas, Avrilia Floratou, Divyakant Agrawal, and Amr El Abbadi. Sphinteract: Resolving Ambiguities in NL2SQL Through User Interaction. PVLDB, 18(4): 1145 - 1158, 2024.
doi:10.14778/3717755.3717772

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ZhaoFuheng/Sphinteract.

1 INTRODUCTION

Given a database D and a natural language question *nl*, the goal of converting Natural Language to SQL queries (NL2SQL, for short) is to find a SQL query Q that answers *nl*. NL2SQL tasks have broad applications in databases [61], geographical information systems [66], healthcare [31], and code generation [11], to name a few. A good quality NL2SQL solution can greatly enhance the productivity of data scientists and engineers, by allowing individuals with various levels of SQL expertise to interact with the database easily. Given its importance in making data analytics easier for users, the problem has received significant attention from multiple communities in both academia and industry [8, 15, 16, 32, 69]. Recent progress in developing pre-trained language models [65] and LLMs [2] has demonstrated strong capabilities for NL2SQL. Using LLMs, [17, 42] has achieved impressive performance on popular NL2SQL benchmarks (such as Spider [64] or WikiSQL [75]).

Although LLMs have brought advancements and showcased strong potential for addressing NL2SQL, there remain several complex challenges to support enterprise-grade NL2SQL [16, 19]. One of the key challenges, and the main focus of this paper, is to identify the *intent* of user's natural language questions accurately, and then generate SQL queries that match the user intent.

Example 1.1. Consider the following question from the GeoNuclearData database in the KaggleDBQA benchmark: **Where is the first `'BWR'` type power plant built and located?**
The database contains a single table `nuclear_power_plants`. The table includes columns such as `Name`, `Latitude`, `Longitude`, `Country`, `ConstructionStartAt`, `OperationalFrom`, etc. Despite the database being simple, directly prompting GPT-4 Turbo [38] with the question and the schema information leads to the following incorrect SQL: `SELECT Country, Name FROM nuclear_power_plants WHERE ReactorType = 'BWR' ORDER BY OperationalFrom ASC LIMIT 1`. There are two immediate ambiguities that appear in the question.

- **How to define the *first* BWR power plant:** There are at least two possible ways of how to define the first power plant. It could be defined since a power plant becomes operational or based on when the construction of the power plant started.
- **Query output:** Although the LLM makes a reasonable choice of picking the Country and the Name of the power plant in the selection clause of the query, it is incorrect when compared to the ground truth query (which expects `Latitude` and `Longitude` as the output) in the benchmark. Thus, it is not clear what are the columns expected in the output of the query.

There are several known causes of ambiguities in an NL2SQL translation. First, in contrast to many popular academic NL2SQL benchmarks [64, 75], real-world databases often contain noisy data values, lack meaningful semantics in their table and column

Proceedings of the VLDB Endowment, Vol. 18, No. 4 ISSN 2150-8097.
doi:10.14778/3717755.3717772

names, and usually do not provide metadata with column specifications [19]. As demonstrated by Example 1.1, without the knowledge about the semantics of which columns to use to determine the first power plant of `'BWR'` type, answering the NL question is difficult. Second, NL2SQL benchmarks have a single correct answer for each NL question. However, in practice, the same question has multiple possible interpretations that may vary from user to user [16]. Thus, understanding the user intent is critical to make sure that the interpretation a user has in mind is used to answer the NL question. For instance, as shown in Example 1.1, knowing whether the user wants `Country, Name` or `Latitude, Longitude` as the output columns of the query is important to answer the NL question correctly. Therefore, unidirectional approaches (i.e., directly providing the answers with no interaction with the user) fail to disambiguate a user's question and do not scale to real-world datasets. Although prior works have also observed such challenges and looked into incorporating user feedback during semantic parsing of SQL queries using pre-trained models (e.g., BERT [27]), solutions are not scalable since they rely on rule-based algorithms that work for a small fragment of the SQL syntax to explain a SQL query [51], require significant effort from the user to reason and edit the SQL query [14, 51], and have severe restrictions on the type of questions that can be asked (for example, [63] only allows asking whether a column should be included in the query or not.)

To address the aforementioned challenges, in this paper, we propose a new interactive framework called Sphinteract [1] that uses LLMs to address the limitations in prior approaches. Motivated by the strong reasoning and code generation capabilities in LLMs, we investigate how LLMs perform with different types of user interactions. We develop new techniques in Sphinteract to iteratively detect ambiguities, ask clarification questions, and incorporate user feedback in generating the user's expected SQL answer. This interactive process allows the LLMs to refine its SQL reasoning and generating process, leading to more accurate and context-aware responses. The framework's key innovation lies in its ability to dynamically adjust NL2SQL answers based on the user's intent, thereby improving utility and reliability. We summarize our overall contributions as follows.

- We propose the Sphinteract framework for NL2SQL tasks. The framework uses three different algorithms for generating questions that are used to solicit feedback. A core ingredient in our techniques is the proposed *Summarize, Review, and Ask* (SRA) paradigm to help LLMs identify ambiguities and generate meaningful clarification questions. Based on the feedback from the user to the clarification questions, the input is used in an iterative fashion to narrow down the ambiguity and generate higher quality SQL statements.
- We conduct a user study with production team engineers and data scientists to understand the different types of ambiguities present in NL2SQL tasks. The resulting diverse high quality queries inform us on the different expectations users have when using NL2SQL tasks, which in turn helps in carefully constructing prompts for the LLM to identify possible ambiguities in a natural language question. The ground truth annotations obtained via our study may be of independent interest as well.
- Our experiments on two challenging datasets, as well as a user study with 11 participants, demonstrate the effectiveness of our approach. Our experimental results show that Sphinteract can increase the execution accuracy by 42.3% on KaggleDBQA and 26.86% on BIRD in the zero-shot SQL setting; and by 30.03% on KaggleDBQA and 32% on BIRD for the few-shot setting in SQL generation by using at most four interactions with the user. Further, these gains are observed consistently across two different LLM models.

[1]The name takes inspiration from the story of the Sphinx, known for posing riddles that required deep thought and understanding to solve.

2 BACKGROUND AND RELATED WORK

2.1 Preliminaries on LLMs

LLMs are trained on large amounts of data, and by using the attention mechanisms [52], these language models can generate high-quality answers based on the input prompt. They have demonstrated strong capabilities in solving complex tasks in mathematics [46], logics [68, 74], and semantics [72].

Few-shot learning. Given a few examples of the form $\{(x_i, y_i)\}$, where x_i is the description of the task (e.g., the natural language question) and y_i is the answer (e.g., the gold/ground truth SQL query), prompting the LLMs with a prefix of "$x_1 \rightarrow y_1; \ldots; x_i \rightarrow y_i$" before asking the question, leads to a more accurate answer compared to when no examples are provided in the prompt. LLMs exhibit good performance when provided with few-shot examples for a wide variety of tasks [9]. A core capability of LLMs is that they follow natural language instructions. When the input prompt contains specific instructions and requirements (e.g., output in JSON format), the models will ensure the generated output satisfies the instructions in the input prompt. A recent study [76] has found models such as GPT-4 strictly follow the instructions around 80% of the time, and this capability increases with larger model sizes.

Separation between generation and understanding capabilities. It has been shown that while LLMs can often provide expert-level generated answers, LLMs consistently need to catch up in the aspects of understanding [58]. As a result, while the LLMs can generate seemingly correct answers to a given question, they may need help understanding what the question is precisely looking for. LLMs limited ability in comprehending user's intent in NL2SQL motivated us to explore methods to overcome this shortcoming.

2.2 Prior Work

2.2.1 NL2SQL. NL2SQL has been an active area of research in both the database and the NLP community for over four decades, dating back to 1980s [20, 56]. The goal is to construct a SQL query that answers the natural language question. As noted in study [33], rule based methods [41, 47] were popular before 2010s. Usually, rule based approaches parse the input question to construct a parsing tree and then construct the output SQL query by applying a set of rules. From the late 2010s, with the rapid advancements in learning-based techniques, NL2SQL tasks are tackled primarily with neural networks or pre-trained models [23, 44, 53, 54, 75]. More recently, LLMs have shown powerful reasoning, domain generalization, and semantic parsing capabilities [36, 73]. ChatGPT, for example, is able to surpass all previous models on NL2SQL benchmarks [34, 64]. We refer the reader to the survey papers [26, 28] for more details. Our investigation into enhancing the NL2SQL pipeline through interactions is not the first of its kind. Earlier rule-based natural language interfaces presented SQL parsing trees to users to solicit feedback. However, these methods implicitly assumed that users possessed intricate knowledge of SQL and could guide the system in modifying the SQL structure. Another previous study [24] presents a method to develop a semantic parser through crowd-sourcing and improving the semantic parser through feedback. When a SQL prediction is incorrect, a crowd worker will fix the incorrect SQL query. The gathered SQL queries are later used to train future models. Having crowd workers fix the incorrect SQL queries is expensive, and due to the inherent ambiguities, workers may still fail to provide the SQL query that may satisfy the user. Prior research [22] has also explored improving SQL generation by LLMs through the use of documentation and data values to enhance domain-specific column understanding by addressing ambiguity about column semantics.

2.2.2 *Self-Debug.*

Our proposed Sphinteract framework has a close relationship to the ability of LLMs in self-debugging, i.e., LLMs fix the incorrect SQL queries based on the provided feedback. As we will show, even the simplest form of user feedback, a binary response, can measurably enhance the quality of the generated SQL responses. It is important to note that our findings do not contradict previous research in the area of self-correction via LLMs [10, 21], which find LLMs cannot improve their answers with feedback generated by the LLMs themselves. Our approach differs fundamentally as it incorporates feedback directly from users, making this external interaction a pivotal aspect of our framework.

2.3 Prompting and Knowledge Graphs

Prompting techniques are at the core of applying LLMs in different domains. Gradient-based prompting optimizations aim to automatically find better prompt templates [43, 67]. However, these optimizations often require access to the model's training data. Another approach is the use of reasoning chains, exemplified by the popular Chain-of-Thought (CoT) technique [57], which encourages the model to process information sequentially before arriving at a final conclusion. Building on similar concepts, researchers have also utilized structured data representations to enhance problem-solving capabilities. These include tree structures [62], graph structures [6], and table structures [55, 74], enabling models to better address and solve complex questions by systematically organizing and analyzing information. Our proposed SRA template aligns with this paradigm, where we direct models to summarize user feedback and review ambiguities.

A recent line of work that also incorporates contextual information uses knowledge graphs and ontologies to encode context [4, 13, 48]. Preliminary results show that ontologies are capable of providing higher accuracy for LLM powered systems. However, we note that building such ontologies from scratch can be expensive. Further, as we will show in our user study, users frequently have different expectations for the same question. Thus, there is also a need to build *personalized* ontologies that capture user preferences. We view answering of clarification questions as an incremental way to build ontologies on-the-fly in a personalized way. A deeper exploration of the interplay between ontologies and user interactions with LLMs is left as an exciting problem for future work.

3 PROBLEM STATEMENT

The overarching goal of this work is to effectively deal with ambiguity for NL2SQL. Since ambiguity is a fundamental characteristic of natural language, managing ambiguities in NL2SQL is crucial for understanding a user's intent, revising the generated queries, and aligning with the user's expectations. To study the type of ambiguities present in NL2SQL, we conducted a thorough user study on sampled questions from the realistic KaggleDBQA bechmark [30] (we present these results in Section 4). Our key idea is to disambiguate user questions by allowing LLMs to interact with the end user via the mechanism of *clarification questions*.

Setup and Problem Statement. In our setup, the user inputs a natural language question nl, and seeks a SQL query Q over the database D, such that $Q(D)$ answers the question nl. In the 0^{th} round, the LLM directly generates an initial prediction Q_0 to answer nl. When the LLM incorrectly answers nl with Q_0, the interaction process starts. In our setup, the 1^{st} round consists of taking the nl, D, the first user feedback CF_1 as input and generate a revised query Q_1. This process continues until the user accepts the most recently generated query or a stopping criterion is met (e.g., a fixed number of rounds). Our goal is to minimize (a function of) the number of interaction rounds n and the *costs* of invoking the LLM while ensuring that each iteration progressively improves the quality of the generated SQL query to correctly answer nl. This goal can be written as Equation 1.

$$min\{\mathcal{V}(n, costs) | \mathbf{1}_Q(LLM(nl, \mathcal{D}, CF_{1,\ldots,n})) = 1\} \tag{1}$$

$LLM(\cdot)$ returns a SQL query Q' as the output, $\mathbf{1}_Q(Q')$ returns 1 if the the execution results for Q and Q' on the underlying database are the same (i.e. the standard execution match metric), and $\mathcal{V}(.,.)$ is a function of the number of interactions and the cost associated with calling the LLM. As an example, a user may want to minimize the number of interactions, which is done by setting $\mathcal{V}(n, costs) = n$.

It is easy to see that when the number of interactions is large enough, any finite SQL query Q, can be found by iteratively asking questions about each token of the query. However, this approach is tedious and does not enhance the overall user experience. Furthermore, this approach also assumes that the user is an expert in SQL. Our focus in this work is to generate high-quality clarification questions and then incorporate the user feedback on these clarification questions to eliminate the ambiguities in natural language questions and help LLMs construct accurate SQL query answers.

Solution Desiderata. Next, we outline several guiding principles based on our discussions with customers of NL2SQL.

1. Clarification questions should be easy to understand. Since the user may not have deep knowledge of SQL, the clarification questions should not contain any SQL specific syntactic content. For instance, data scientists are proficient with Pandas and may have some familiarity with SQL, but may not know the intricacies of SQL itself. The clarification questions should contain pragmatic language that most people are capable of understanding and reasoning about.

2. Answering clarifications questions should be simple. It is important to keep in mind that seeking clarifications can be burdensome to the user. We believe it is best to use a multiple-choice format. When the highly likely answers are directly included in the multiple choices, the user can simply select the desired answer. Only when none of the choices are correct, the user may have to provide free-form feedback by typing. Furthermore, the clarification questions and answers are tailored specifically to the user's anticipated SQL queries, rather than being derived from an open domain. We elaborate on this aspect in the evaluation section.

3. Avoiding unnecessary interactions. While bounding the number of interactions to a fixed number may seem natural, NL questions have different ambiguity and difficulty levels, and thus an *a priori* fixed number of interaction rounds may not be well suited for all questions. Moreover, there are other sources of errors such that LLMs may still not achieve a 100% execution accuracy when all ambiguities have been resolved for all questions. For example, LLMs sometime may hallucinate or fail to correctly reason about complex database schema. Due to the complex nature of how LLMs handle ambiguities in NL2SQL tasks, we investigate both the approach of using fix a interaction budget and the approach of stopping when no remaining ambiguities are detected.

4. Show the user queries that are executable. Since the user has access to the database, it is important to make sure that the SQL query shown to the user is executable. This also aligns with the third desiderata since correcting syntactic errors in SQL queries potentially reduces interactions with the user.

4 USER STUDY AND MOTIVATION

This section presents our user study over 64 randomly sampled questions from KaggleDBQA [30], which has eight cross-domain databases with multiple tables. The goal is to identify common challenges in writing SQL answers and to validate our hypothesis that ambiguities are ubiquitous in NL2SQL tasks, even for well-established NL2SQL benchmarks.

4.1 Human Annotations and Analysis

We asked seven computer scientists who have deep knowledge of SQL to write the SQL query for a given NL. We made sure that each question was answered by at least three experts. Each user was given between 16 to 20 questions to annotate. In the user study, we specifically asked the experts to "provide multiple SQL statements for questions that you think are ambiguous" and they could also provide comments on why they think the question is ambiguous. The database schema information along with the questions were provided to the experts. All human experts performed the study independently. In addition to the seven experts, we also appended the gold SQL queries from KaggleDBQA benchmark, as they reflect the interpretation of the benchmark creators. In the end, we executed all queries on the underlying database, and if a SQL query is not valid (i.e. not executable), the authors fixed the errors in the SQL query (e.g., misspelled column names) while still keeping the query aligned with the logic of the original invalid SQL query.

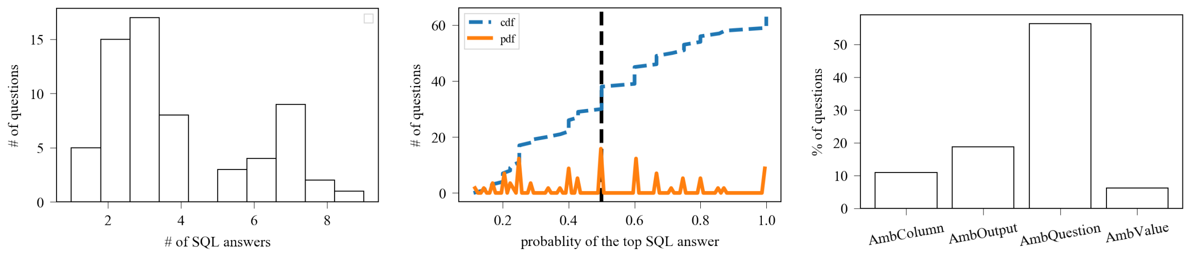
Analysis. We find that the number of SQL answers to each question follows a skewed distribution. Only a few questions have a single unique SQL answer, and many questions have several different SQL answers. As shown in Figure 1 (a), the x-axis is the number of unique SQL answers, and the y-axis is the number of questions. We consider two SQL queries to be the same if they give the same output table when executed on the same database. Only 8% of the questions (5 out of 64) have a single SQL answer (i.e. all SQL queries written for the question map to the same output table).

In some cases, a question may have different interpretations, but there might be an implicit consensus on the most likely answer. As a result, we looked into the degree of agreement for the most popular SQL answer (the top SQL answer) for each question in Figure 1 (b). The x-axis is the likelihood of the top SQL answer, and the y-axis is the number of questions with that likelihood. For example, if a question has 3 SQL answers and two of them give the same output table, then the top SQL answer has a likelihood of 2/3. We find that roughly 50% of the questions have a majority consensus, while only 5 questions have a unique answer, confirming that ambiguity is ubiquitous in NL2SQL tasks. Consequently, for the same NL question, two different users may have completely different expectations of the output.

5 AMBIGUOUS TYPES

Next, we classified the ambiguities found in the user study, taking expert's comments into consideration. In particular, we identified four different types of ambiguities: AmbColumn, AmbOutput, AmbQuestion, and AmbValue. Note that each type of ambiguity is not exclusive. The distribution of these ambiguity types is depicted in Figure 1 (c). AmbColumn happens when the entities in the natural language question do not have a clear mapping to the database schema. The main causes for AmbColumn are unclear semantic of the columns (e.g., columns with dummy names: col1, col2, ...) and when at least two columns share similar semantics (e.g., locations and areas). AmbOutput occurs when the natural language question does not specify the expected format or ordering of the output table. To accurately answer the user's query, it is essential to understand specifics such as the number of columns to select, whether to include aggregates or present information in a specific format and if duplicates in the output are acceptable. AmbQuestion happens often since ambiguity is intrinsic to natural language. For example, there may be lack of clarity on how to compute an aggregate or the NL question is vague (such as "tell me something cool about the data"). Based on different interpretations of the question, the SQL queries can be drastically different. AmbValue occurs when there is uncertainty about the appropriate predicate values to use. The predicates referenced in the question may differ from their physical storage formats in the database, such as being encoded as numbers, or phrases. We have also listed example questions and corresponding SQLs in Table 1.

Example 5.1. Continuing the running example from Example 1.1, we observe that the NL question **Where is the first `'BWR'` type power plant built and located?** exhibits AmbQuestion since there are two valid interpretations of how to define the first power plant and thus, making progress on generating the query requires clarification about the definition. The question also has AmbOutput since the *nl* can be interpreted as asking for the `Country` and `Name` or the `Latitude` and `Longitude` of the power plant as the output.



(a) Cardinality of each Question **(b) Distribution of Top Answers** **(c) Ambiguous Types**

Figure 1: Analysis on the User Study.

Table 1: Example Questions and the corresponding SQLs for each Ambiguity Types. The different SQL clauses employed by users to answer the same question are highlighted.

Type	NL Question and SQLs
AmbColumn	For award winners, which position has the most hall of fame players?
	SELECT notes FROM player_award AS p JOIN hall_of_fame AS h ON p.player_id=h.player_id WHERE inducted='Y' GROUP BY notes ORDER BY COUNT(*) DESC LIMIT 1;
	SELECT category FROM player_award as p JOIN hall_of_fame as h ON p.player_id = h.player_id WHERE inducted = 'Y' GROUP BY category ORDER BY COUNT(h.player_id) DESC LIMIT 1;
AmbOutput	What is the torrent download statistics for each release year?
	SELECT groupYear, SUM(totalSnatched) AS total FROM torrents GROUP BY groupYear ORDER BY groupYear;
	SELECT SUM(totalSnatched) FROM torrents GROUP BY groupYear;
AmbQuestion	Which artist/group is most productive?
	SELECT artist, COUNT(*) AS total FROM torrents GROUP BY artist ORDER BY total LIMIT 1;
	SELECT artist FROM torrents GROUP BY artist ORDER BY SUM(totalSnatched) DESC LIMIT 1;
AmbValue	Show all fires caused by campfires in Texas?
	SELECT * FROM Fires WHERE state = 'TX' and STAT_CAUSE_DESCR = 'Campfire';
	SELECT * FROM Fires WHERE state = 'TEXAS' and STAT_CAUSE_DESCR LIKE '%Campfire%';

6 SPHINTERACT

To address the ubiquitous ambiguities in NL2SQL tasks, we propose the Sphinteract framework, which consists of multi-round interactions with the end user. The implementation of the interaction process can be separated into two parts: (*i*) generate a SQL query to show the user (subsection 6.1), and (*ii*) gather user feedback (subsection 6.2). The first part is responsible for learning from earlier mistakes and incorporating user feedback to generate high-quality SQL answers. The second part involves communicating with the end user to gain clarity on the user's question and the user's intent.

Sphinteract framework workflow is shown in Figure 2. In step 1 and step 2, Sphinteract generates an executable SQL query based solely on the given *nl* question and the database schema. After executing this SQL query on the database in step 3, it presents the output and the query to the user who may inform Sphinteract that the proposed query does not satisfy their expectations (step 4). In Section 6.2.1, we discuss how this simple feedback of yes or no alone can be used to help improve future predictions. In Section 6.2.2 and 6.2.3, we introduce the *SRA* prompt to seek feedback with a multiple-choice question, depicted in step 5. In step 6, the collected user feedback is then effectively used to resolve ambiguities in answering the user's question. Step 7 and 8 presents the revised SQL query and output of its execution on the database to the user. The process continues until an ending criterion is met.

6.1 SQL Generation

There are many different choices in prompting the LLMs to answer a SQL query, and there exist different prompting techniques to improve single round SQL generation. In Sphinteract, we adopt the state-of-the-art DAIL-SQL prompting template [18]:

```
Complete sqlite SQL query only and with no explanation.
/* Given the following database schema: */
{schema}
/* Answer the following with no explanation:{question} */
SELECT
```

DAIL-SQL consists of instructions, database schema, and questions. The columns of each table in the schema are encoded as a comma separated list. DAIL-SQL prompts have been carefully crafted to identify the best format for presenting the NL questions and the schema information. We note that in DAIL-SQL, only the schema level information (such column names and PK-FK information) is provided in the prompts[2]. The prefix "Complete sqlite SQL query only and with no explanation" is a specific rule introduced by OpenAI's official Text-to-SQL demo [37], which has been found to improve the quality of generated SQL across different models consistently (across different language models) [17]. In practice, an LLM may generate invalid SQL queries (e.g., misspelled columns and missing table prefixes). When a SQL exception is raised from

[2]Data values are not provided due to privacy and security concerns, as well as prompt window size constraints.

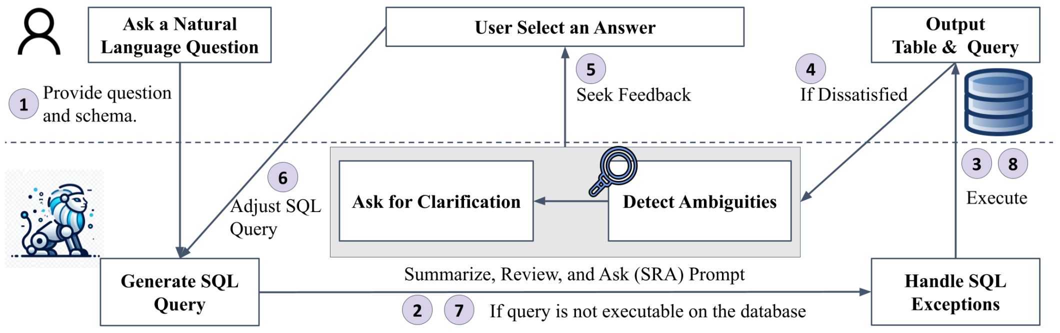


Figure 2: Demonstration of the workflow in SPHINTERACT .

executing the generated code, we give the LLM another chance to fix the error by providing the invalid query, the exception error message, and the schema.

6.2 Seeking User Feedback

In this section, we describe three techniques to seek feedback from the user, starting from the basic approach that will form our baseline to increasingly advanced strategies.

6.2.1 Simple User Feedback. Taking inspiration from the simple feedback approach in [10] for code generation, we also investigate if the simple feedback of whether the generated SQL is correct or incorrect according to the user can be used to improve the overall quality of the generated SQL answers. If a generated SQL query is incorrect, the incorrect query, along with incorrect queries from prior rounds, are also provided in the prompt, and the LLMs are instructed to generate a revised SQL query.

We highlight the core difference between the simple feedback in our work and [10]. In SPHINTERACT, we operate under the assumption that users can offer feedback on whether the generated SQL queries meet their expectations; hence, the feedback is conditioned on the user's expectations. In contrast, the approach taken in [10] involves obtaining feedback directly from the model by having it determine whether the generated SQL query correctly answers the given question. Since the feedback in [10] is independent of the user's expectations, there may be discrepancies: the model may deem a SQL query correct based on its own interpretation of the question, while the user might have a different interpretation in mind.

6.2.2 Asking For Clarifications. While the simple feedback approach is very easy to implement, it may be sub-optimal. The simple feedback, in essence, is searching for the correct SQL query (the one that satisfies the user's expectation) one by one via a guided search. To accelerate the steps in finding the correct SQL query, we believe that asking clarification questions is more efficient. We employ a multiple-choice format for clarification questions, consistently including an option for users to provide free-form input, which is a standard practice in soliciting user responses [25]. In addition to logging incorrect SQL queries, we also prompt the LLM to identify ambiguities within the question and then pose a clarification question to the user. Once we receive this clarifying feedback, we instruct the LLM to generate a new SQL query that accurately reflects the user's input and aligns with their feedback.

The prompt for both generating the SQL answer and generating the clarification questions takes incorrect queries and the user feedback on the incorrect queries as input. In the SQL generation stage, using this information in the input prompt helps LLMs generate better SQL answers by learning from past mistakes and user feedback. In the clarification question generation stage, including this information in the input prompt helps the LLMs to effectively pinpoint the remaining ambiguities and prevent the model from repeating previously asked clarification questions.

In addition, we introduce a new technique called 'Summarize, Review, and Ask' (SRA), which enables LLMs to summarize information from previous interactions, assess and review the remaining ambiguities, and ask targeted clarification questions to the user in a streamlined process. We also include four ambiguity types identified through our user study. During the review step, we instruct the LLM to reason about any unresolved ambiguities based on the four ambiguity types. Finally, we ask the LLM to generate a question that will help resolve one of the ambiguities. The algorithm obtained with the use of the SRA prompt template will be referred to as the CQs algorithm. The SRA template is shown below:

```
1. Summarize the information that is clear based on the
   answrs to previous clarification questions and
   incorrect queries.

2. Evaluate whether AmbQuestion, AmbColumn, AmbOutput,
   and AmbValue remain in formulating a SQL query to
   answer the QUESTION, considering each category
   individually.

3. Ask a multiple-choice clarification question to
   clarify the remaining ambiguities and help you find
   the correct SQL query.
```

Since we want the generated clarification question to be in a specific format (a multiple choice question with three closed-ended options and one open-ended option), we provide static few-shot demonstration examples in the prompt to help the LLM generate the questions in the correct format.

6.2.3 Early Stopping (ES). Although ambiguity is one of the key factors causing LLMs to struggle with NL2SQL tasks, they may still fail to correctly answer a question, even after all ambiguities are addressed. For instance, LLMs may fail to focus on the user feedback or fail to use the correct SQL keywords due to its inherent limitation in answering complex questions. When all the ambiguities have been addressed, the interactions between LLMs and the user may not be very helpful. Therefore, we add an early stopping (ES) instruction for the LLM. Algorithm 1 shows the algorithm steps with ES. If we remove the ES instruction from the prompt, we get the CQs algorithm from the previous subsection.

The early stopping technique offers two key advantages over the CQs introduced in the previous section. First, reducing the number of interactions directly reduces the dollar cost. As we will show in the experimental section, with early stopping, it is possible to obtain better improvement in accuracy per dollar as compared to CQs. Second, the CQs algorithm has the disadvantage that LLMs can continue to generate clarification questions (owing to their *stochastic parrot* [5] nature where LLMs may continue to generate output without truly understanding the task) even if all ambiguities have been resolved. This is because there is no explicit instruction to stop asking questions when the model thinks no ambiguities are left. However, since each clarification question comes with a free-form feedback option, the user has the option to point the model towards its mistakes (such as a hallucinated table name). Thus, there is an inherent tradeoff between keeping the interactions going to improve accuracy (at the cost of being expensive) versus ending the interactions as soon as possible to keep monetary costs low (but potentially sacrificing accuracy). We report the empirical performance of the two methods in the experimental evaluation (cf. Section 7.3.1).

Algorithm 1 Asking Clarification Questions and Early Stop (CQs & ES)

```
Require: Database schema $schema$, $nl$, Temperature $T = 0$, Number of rounds $n$, User feedback $user(.)$ that returns true or false.
1:  $IncorrectSQLs \leftarrow [], Feedbacks \leftarrow []$
2:  $p$ = SQL_prompt($nl, schema$)
3:  $SQL$ = LLM($T, p$)                    ▷ get the initial LLM prediction
4:  for $i = 1$ to $n$ do
5:      $OUT$ = execute($SQL, D$)          ▷ execute the SQL query
6:      if $user(OUT, SQL)$ then
7:          return
8:      else
9:          $IncorrectSQLs$.Append($SQL$)
10:     end if
11:     ▷ use SRA prompt to recover CQs algorithm
12:     $p \leftarrow$ SRA_with_ES($nl, schema, IncorrectSQLs$)
13:     $MCQ \leftarrow$ LLM($T, p$)          ▷ MCQ generated by the LLM
14:     if $MCQ$ is none then
15:         return                       ▷ Early termination if using ES instruction
16:     end if
17:     $Feedback \leftarrow$ feedback from the user for $MCQ$
18:     $Feedbacks$.Append(($MCQ, Feedback$))
19:     $p$ = SQL_prompt($nl, schema, IncorrectSQLs, Feedbacks$)
20:     $SQL$ = LLM($T, p$)        ▷ get a new SQL prediction from the LLM
21: end for
22: return
```

Example 6.1. Continuing our running example from Example 1.1, Sphinteract with GPT-4-Turbo correctly detects AmbOutput. By inspecting the LLM generated output of the SRA prompt, we see that the LLM reasons that the question *does not explicitly state whether the country or the specific geographical coordinates (Latitude and Longitude) are desired for "location."*.

Thus, in the first interaction, the MCQ generated is: **What do you mean by 'located' in the context of finding the power plant?** with the following options:

Ⓐ The country where it is built
Ⓑ The Latitude and the Longitude
Ⓒ The name of the power plant and the country
Ⓓ Others (please specify).

At this point, the user selects option Ⓑ (based on the ground truth). After taking the feedback into account, Sphinteract generates the SQL query: `SELECT Latitude, Longitude FROM nuclear_power_plants WHERE ReactorType = 'BWR' ORDER BY OperationalFrom ASC LIMIT 1`.

In the next iteration, Sphinteract correctly detects that AmbOutput has been resolved but AmbQuestion still remains. By inspecting the SRA prompt output, we find that the LLM reasoned that *there might be ambiguity in what constitutes the "first" BWR type power plant. This could refer to the earliest "ConstructionStartAt" date or the earliest "OperationalFrom" date.* Thus, for the second interaction, the MCQ is: **What does 'first' refer to in the context of the BWR type power plant being built and located?** with the following options:

Ⓐ The power plant whose construction started first.
Ⓑ The power plant that became operational first.
Ⓒ The power plant that has the oldest update date.
Ⓓ Other (please specify).

The user selects option Ⓐ and with this feedback, the system generates the correct query which matches the ground truth in the benchmark: `SELECT Latitude, Longitude FROM nuclear_power_plants WHERE ReactorType = 'BWR' ORDER BY ConstructionStartAt ASC LIMIT 1`.

7 EVALUATION

In this section, we evaluate our framework over the three proposed algorithms and their corresponding prompts. For all experiments, we assume the number of interactions between LLMs and the user is at most four rounds, and hence, at most, five SQL queries are generated. In particular, we seek to answer the following questions:

- **Q.1** What is the performance of the three feedback techniques for zero-shot NL2SQL?
- **Q.2** What is the performance of the three feedback techniques for few-shot NL2SQL?
- **Q.3** Do our proposed techniques generalize to multiple LLMs?
- **Q.4** What is the tradeoff between the number of interactions and the improvement in accuracy?
- **Q.5** What are the qualitative feedback and quantitative metrics about effectiveness and usability of the system?

7.1 Data Source and Metrics

Our evaluation was conducted on two real-world relational databases designed for NL2SQL tasks: (i) KaggleDBQA [30] (referred to as

Table 2: The zero shot evaluation results on Kaggle and BIRD with the state-of-the art models. Baseline is the execution accuracy obtained using the DAIL-SQL system.

Model	Source	Method	Baseline	1 Interaction	2 Interactions	3 Interactions	4 Interactions
GPT-3.5 Turbo	Kaggle	Simple	26.44%	28.37% (+1.93%)	30.29% (+3.85%)	30.77% (+4.33%)	30.77% (+4.33%)
		CQs	26.44%	39.90% (+13.46%)	46.63% (+20.19%)	50.00% (+23.56%)	53.85% (+27.41%)
		CQs & ES	26.44%	37.50% (+11.06%)	39.90% (+13.46%)	41.83% (+15.39%)	41.83% (+15.39%)
	BIRD	Simple	28.03%	29.01% (+0.98%)	29.27% (+1.24%)	29.27% (+1.24%)	29.40% (+1.37%)
		CQs	28.03%	31.88% (+3.85%)	35.27% (+7.24%)	38.20% (+10.17%)	39.63% (+11.60%)
		CQs & ES	28.03%	31.68% (+3.65%)	33.37% (+5.34%)	33.70% (+5.67%)	33.83% (+5.80%)
GPT-4 Turbo	Kaggle	Simple	29.33%	34.62% (+5.29%)	35.10% (+5.77%)	35.58% (+6.25%)	36.54% (+7.21%)
		CQs	29.33%	53.37% (+24.04%)	64.90% (+35.57%)	70.19% (+40.86%)	71.63% (+42.30%)
		CQs & ES	29.33%	52.40% (+23.07%)	58.17% (+28.84%)	59.62% (+30.29%)	59.62% (+30.29%)
	BIRD	Simple	32.07%	34.62% (+2.55%)	35.20% (+3.13%)	36.05% (+3.98%)	36.38% (+4.31%)
		CQs	32.07%	45.70% (+13.63%)	52.74% (+20.67%)	56.32% (+24.25%)	58.93% (+26.86%)
		CQs & ES	32.07%	45.96% (+13.89%)	50.46% (+18.39%)	52.22% (+20.15%)	53.00% (+20.93%)

Kaggle henceforth for brevity); and (ii) BIRD [34]. The Kaggle benchmark contains 272 questions covering eight databases in different application domains. On average, each database contains 2.3 tables and 280K rows. Recall that we have used 64 questions for the user study, as shown in Section 4. As a result, we perform the evaluation on the remaining 208 questions. The BIRD benchmark contains 1534 questions in its development set. On average, each database has 7.3 tables and 549K rows. For all experiments in this section, we assume each NL2SQL question contains a natural language question, a corresponding SQL answer (gold SQL), and the database schema.

Metrics. The standard evaluation metric for NL2SQL benchmarks is the execution accuracy. We also compute the precision and recall when looking at the columns and tables present in the predicted SQL query and the gold SQL. Such featurization of the SQL queries is frequently used for measuring query similarity [29]. Monetary cost metrics will be presented as the average cost (in USD cents) of answering a question in the benchmark using a particular algorithm.

7.2 Experiment Configurations

7.2.1 LLMs Configurations. We evaluate our proposed framework on OpenAI GPT models [38]. In particular, we showcase the experimental results using the popular GPT-3.5 Turbo (gpt-3.5-turbo) and GPT-4 Turbo (gpt-4-turbo-preview) models. In the API call, we set the temperature to 0. For all experiments in this paper, we also include the foreign-key-primary-key (FK-PK) constraints in the database schema.

7.2.2 Baselines and Prompt Details. We use the state-of-the-art DAIL-SQL [18] system as the baseline in all of our experiments. We omit comparing against baselines that involve fine-tuned models (such as CHESS [50]) or contextualizing prompts with non-standard information (i.e., beyond schema information and few-shot examples). Such approaches can be considered orthogonal to ambiguity resolution and are limited by security, cost, and updates concerns as we discuss in Section 7.5.1.

Unlike the Kaggle benchmark, BIRD questions come with hints from the benchmark creators that are useful in generating the SQL queries. Since providing hints can only make the NL2SQL problem easier and to keep the experiment configuration consistent with Kaggle, unless specified otherwise, we do *not* add hints to the prompt. However, to aid comparison with solutions that use the hints, we present results of our framework on BIRD but with hints added in the prompts in Section 7.4.4. For all our experiments, we cap the number of interactions to four.

7.2.3 Feedback Oracle. We employ an oracle to simulate the interactions in our framework, a standard method that has been used by prior works [3, 7, 12, 40, 59]. Previous works have also demonstrated that LLMs are capable of answering multiple-choice questions [45] and understanding the logic behind SQL queries [74]. The oracle takes the clarification question and the gold SQL query as input, and then outputs the feedback to clarify the ambiguities. To ensure highly accurate feedback is generated, we leverage few-shot prompting (eight examples) coupled with the chain-of-thought technique on the state-of-the-art GPT-4o model [39].

7.2.4 Few Shot Setting. Few shot examples are used in generating the clarification questions, feedback, and, later in this section, for SQL generation. We use eight static few shot examples for generating the clarification questions. The few-shot examples used in generating SQL queries for the simple feedback algorithm are also static, and we use the same examples as [10]. The few-shot examples used for the SQL generations in CQs and CQs & ES are selected based on the cosine similarity between the vector embedding[3] of the *nl* asked by the user and the vector embedding of question *nl* in the question bank. The question bank for BIRD is the training set provided by the benchmark, and question bank for Kaggle are the 64 questions that were used for the user study in Section 4.

7.3 SQL Zero Shot Experiments

In this section, we evaluate the performance of simple feedback (Simple), asking clarification questions (CQs), and stopping early

[3]For our experiments, we used the text-embedding-ada-002 embedding model from OpenAI.

when no ambiguities remain (CQ & ES) but without providing any examples in the SQL generation prompt (i.e., the zero shot setting).

As shown in Table 2, providing feedback through MCQs to answer clarification questions significantly increases the overall execution accuracy of LLMs. The CQs algorithm (Algorithm 1 with SRA prompt), which seeks the user's feedback to the clarification question, achieves 71.6% and 58.9% execution accuracy on Kaggle and BIRD, respectively, using GPT-4 Turbo. The execution accuracy is improved by 42.3% and 26.9% compared to the execution accuracy with no interactions. This is consistent with our expectations since addressing the different types of ambiguity helps LLMs better understand the user's intent. On the other hand, the simple feedback consistently yields lower accuracy. This is because the binary feedback does not directly resolve ambiguities. The simple feedback mechanism, when using both GPT-3.5 Turbo and GPT-4 Turbo models, only slightly improves the quality of the generated SQL query after the first round of interaction, and oftentimes, no further improvements are made beyond the second round.

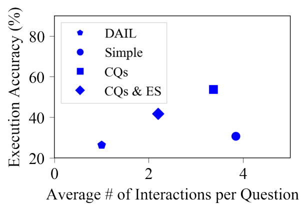
The incremental execution improvements decrease as the number of interactions increases. While many NL questions have low levels of ambiguity, some questions can be hard to understand and require much more interactions to clarify. The CQ & ES algorithm, which halts processing when the LLM determines all ambiguities have been resolved in translating a natural language question to an SQL query, exhibits mixed effectiveness. It requires less round of interactions and less accuracy improvements. Despite incorporating the ambiguities types into our proposed SRA prompt, we noted that LLMs may not fully identify all ambiguities. For example, LLMs occasionally overlook the number of columns selected, a category we term AmbOutput. Consider a case where the user seeks the 'average weight for each position'. The correct SQL query only selects the average weight. However, LLMs often erroneously include both the average weight and other fields (e.g., position) in the output table, failing to satisfy the user's expectation.

Table 3: The efficiency metric and the cost (in cents) per question of the three methods (see Section 7.3.1).

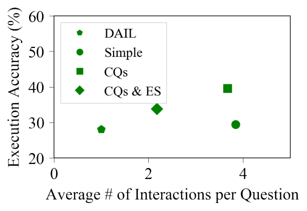
Model	Algorithm	Kaggle		BIRD	
		Efficiency	Cost	Efficiency	Cost
GPT-3.5 Turbo	Simple	1.52	¢.5	0.48	¢1.2
	CQs	11.57	¢2.6	4.36	¢4.0
	CQs & ES	12.93	¢1.8	4.92	¢2.5
GPT-4 Turbo	Simple	2.72	¢0.21	1.64	¢4.2
	CQs	23.20	¢8.2	12.61	¢12.8
	CQs & ES	21.02	¢7.4	15.73	¢9.9

7.3.1 *Trade-offs.*

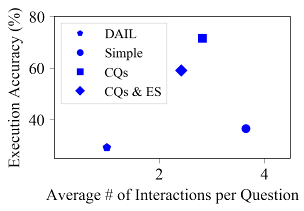
So far, we have only looked at the improvement in accuracy for the three methods. In this section, we study the accuracy improvements compared to the number of interactions. To align with this objective, we introduce the new *efficiency metric* which is defined as the ratio of the incremental improvement in execution accuracy (compared to no interaction) and the average number of interactions for questions. This efficiency metric closely relates to our optimization objective shown in Equation 1. A higher



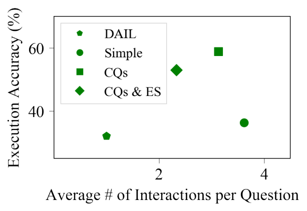
(a) GPT-3.5 Turbo Kaggle



(b) GPT-3.5 Turbo BIRD



(c) GPT-4 Turbo Kaggle



(d) GPT-4 Turbo BIRD

Figure 3: Tradeoffs between the total execution accuracy and the average number of interactions.

value of this efficiency metric indicates that the algorithm is able to converge more rapidly (i.e. in fewer interactions) to the correct answer. In Figure 3, we observe that while CQs always achieve the highest execution accuracy, CQs & ES uses less number of interactions to score higher execution accuracy than Simple Feedback.

Table 4: Precision and recall for the zero shot setting after all interactions. The numbers in bracket show the improvement compared to the Baseline.

Model	Dataset	Algorithm	Precision & Recall
GPT-3.5 Turbo	Kaggle	Baseline	0.80 & 0.78
		Simple	0.80 (+0%) & 0.76 (-2%)
		CQs	0.91 (+11%) & 0.88 (+10%)
		CQs & ES	0.88 (+8%) & 0.84 (+8%)
	BIRD	Baseline	0.88 & 0.82
		Simple	0.87 (-1%) & 0.77 (-5%)
		CQs	0.91 (+3%) & 0.87 (+5%)
		CQs & ES	0.90 (+2% & 0.86 (+4%)
GPT-4 Turbo	Kaggle	Baseline	0.77 & 0.85
		Simple	0.80 (+3%) & 0.85 (0%)
		CQs	0.90 (+13%) & 0.92 (+7%)
		CQs & ES	0.90 (+13%) & 0.91 (+6%)
	BIRD	Baseline	0.88 & 0.88
		Simple	0.88 (+0%) & 0.89 (+1%)
		CQs	0.94 (+6%) & 0.93 (+5%)
		CQs & ES	0.93 (+5%) & 0.92 (+4%)

As shown in Table 3, the more powerful GPT-4 Turbo model consistently achieves higher efficiency compared to the weaker GPT-3.5 Turbo model. We also find that the CQs and CQs & ES algorithm always achieve higher efficiency (using less number of interactions or costs to improve execution accuracy) than the simple feedback algorithm on both GPT-3.5 Turbo and GPT-4 Turbo models. Table 3 also shows the dollar cost incurred per question. The first observation is that CQs & ES is cheaper (but achieves lower total improvement in accuracy) compared to CQs, which is in line with our expectation since CQs requires more interactions as we had hypothesized in Section 6.2.3. Interestingly, the simple feedback is (up to) an order of magnitude cheaper compared to the other two algorithms. This is because simple feedback generates no clarification questions and thus, no LLM output tokens are used[4], making the LLM calls significantly cheaper. Table 4 shows the precision and recall metrics based on the exact match of columns and tables. Similar to execution match accuracy, both precision and recall improve for CQs and CQs & ES which signifies that as more and more interaction happen, the correct set of columns and tables are incorporated into the generated query and the incorrect columns and tables are removed. The reader may see that compared to the baseline, the precision and recall for simple feedback does not experience much improvement (and sometimes decreases). Upon deeper investigation, we observed that with just the binary feedback of whether the LLM generated query is correct or not, the LLM tends to take more *risk* in subsequent rounds and generates queries with more variety by incorporating different tables and columns. For example, for the question **What is the rule of playing card "Benalish Knight"?** from the `card_games` database in the BIRD benchmark, the (incorrect) baseline query generated is `SELECT text FROM cards WHERE name = 'Benalish Knight'`. When using simple feedback, the query generated in the next iteration is `SELECT text FROM rulings WHERE uuid = (SELECT uuid FROM cards WHERE name = 'Benalish Knight')`. Note that the LLM adds the `rulings` table into the query. However, CQs gets the feedback from the user that the output required is the legality of the card in different formats. When the LLM takes this feedback, the query generated in the next round (which matches the ground truth) is `SELECT format FROM legalities JOIN cards ON legalities.uuid = cards.uuid WHERE cards.name = 'Benalish Knight'`, correctly identifying that the format comes from the `legalities` table.

We conclude this section by highlighting the benefits of the SQL exception handler. Inspecting the logs of the CQs algorithm on Kaggle, we observed that GPT-3.5 turbo and GPT-4 turbo are able to fix 95% of the SQL exceptions after providing the invalid query, exception message, and the schema.

7.4 SQL Few Shot Experiments

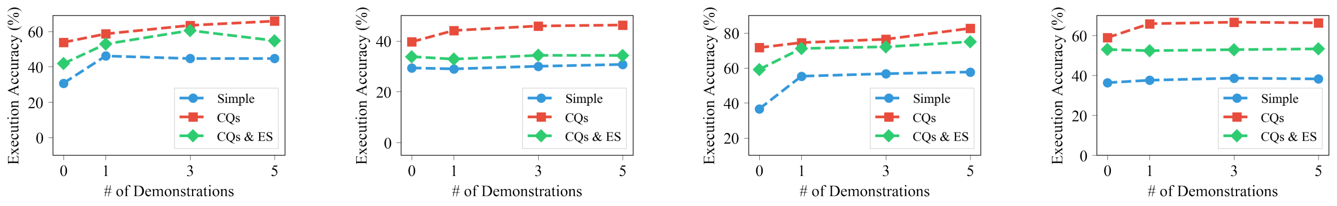
The capabilities of language models can be increased with in-context learning [9]. The primary distinction between the few-shot and zero-shot experiments lies in the inclusion of several examples during the SQL generation phase. Previously, we hypothesized that the primary reason for incorrect SQL responses, even with substantial feedback, is the failure of language models to construct the appropriate SQL structure. Introducing examples during the SQL generation phase should enhance the quality of the SQL query.

[4]Output tokens cost $3 - 4\times$ more than the input tokens for GPT models.

7.4.1 Impact of Number of Examples on Accuracy. As shown in Figure 4, the x-axis depicts the number of examples added into SQL generations, and the y-axis is the total execution accuracy (note the different scales) after all rounds of interactions. The few-shot examples for the simple feedback are static, and the few-shot examples for CQs and CQ & ES are selected based on the question similarity with questions in question bank. From Figure 4, we can observe that by increasing the number of examples, the total execution accuracy often improves. For example, the execution accuracy of CQs always increases. For the simple feedback and CQs & ES, the accuracy may slightly decrease or plateau when increasing from 3 to 5 examples, as shown in Figure 4 (a)-(c). This phenomenon aligns with previous NL2SQL research, which suggests that a greater number of examples does not necessarily enhance accuracy due to the limited in-context learning capabilities of LLMs [17]. In addition, the ability of the GPT-4 Turbo model to self-correct SQL queries when given some examples is impressive. The simple feedback algorithm is able to achieve 3% - 8% execution accuracy improvements, CQs is able to achieve 25% - 32% execution accuracy improvements, and CQs & ES is able to achieve 18% - 25% execution accuracy improvements. These observations demonstrate that even the simplest form of feedback can lead to substantial improvements in the quality of SQL answers through in-context learning by LLMs. Although CQs & ES always has lesser execution accuracy improvement compared to CQs, CQs & ES involves fewer interactions with the user and thus, has a smaller dollar cost. We discuss the efficiency and the cost of these algorithms in the next section.

7.4.2 Improvements through User Interactions. In Table 5, we show the improvements achieved by LLMs through rounds of interactions and their corresponding efficiency for the few-shot experiments. When employing CQs, the performance enhancements observed are quite significant. Specifically, the GPT-3.5 Turbo model registers ~25% improvement in execution accuracy on Kaggle and ~18% improvement on BIRD compared to the strong baseline DAIL-SQL. Meanwhile, the GPT-4 Turbo model shows even more significant accuracy improvements, achieving around a 30% improvement on Kaggle and on BIRD compared to the DAIL-SQL baseline. The highest execution accuracy for Kaggle is 82.7%, and for BIRD is 66.7% (see 3-shot and 5-shot GPT-4 Turbo in Table 5). Even with the most basic form of binary feedback provided over a maximum of four rounds of interaction and utilizing a static set of five examples during SQL generation, simple feedback still achieves a respectable 38.7% accuracy on BIRD. This highlights that even the most basic form of interactions can markedly improve outcomes in complex tasks. Moreover, asking clarification questions lead to much higher accuracy improvements. By engaging in more detailed conversation with the user, models can refine their understanding and address the specific needs of each *nl* more effectively.

7.4.3 Efficiency, Precision, and Recall. Table 5 also shows the efficiency and the precision/recall metrics. CQs consistently achieves the highest execution accuracy improvements and often delivers the best efficiency, albeit with the highest monetary cost. CQs & ES, while slightly less effective in accuracy improvements, requires lower costs. Simple feedback is considerably less efficient compared to CQs and CQs & ES. For precision and recall, we observe the same trend as in the zero-shot setting where simple feedback experiences



(a) GPT-3.5 Turbo Kaggle **(b) GPT-3.5 Turbo BIRD** **(c) GPT-4 Turbo Kaggle** **(d) GPT-4 Turbo BIRD**

Figure 4: The evaluation of incorporating in-context learning for SQL generation.

Table 5: The few shot evaluation on Kaggle and BIRD after all interactions. Cost denotes the average cost (in cents) per question. The numbers in brackets report the accuracy improvement compared to the *Baseline* method.

Few shot	Source	Method	GPT-3.5 Turbo	Prec. & Recall	Efficiency	Cost	GPT-4 Turbo	Prec. & Recall	Efficiency	Cost
3-shot	Kaggle	Baseline	41.8%	.84 & .87	-	¢.1	50.5%	.86 & .88	-	¢.5
		Simple	44.7% (+2.9%)	.84 & .87	1.3	¢.7	56.7% (+6.2%)	.87 & .88	3.5	¢2.0
		CQs	63.5% (+21.7%)	.91 & .92	11.8	¢2.4	76.4% (+25.9%)	.96 & .95	19.5	¢6.8
		CQs & ES	60.6% (+18.8%)	.89 & .91	16.9	¢1.8	72.1% (+21.6%)	.93 & .94	21.7	¢6.0
	BIRD	Baseline	28.0%	.87 & .85	-	¢.3	34.7%	.89 & .88	-	¢1.1
		Simple	30.0% (+2.0%)	.88 & .86	0.7	¢1.6	38.7% (+4.0%)	.90 & .89	1.6	¢4.9
		CQs	46.0% (+18.0%)	.92 & .90	6.9	¢4.4	66.7% (+32.0%)	.94 & .92	16.3	¢12.9
		CQs & ES	34.4% (+6.4%)	.90 & .88	5.1	¢2.9	52.9% (+18.2%)	.93 & .92	14.1	¢10.5
5-shot	Kaggle	Baseline	41.8%	.83 & .88	-	¢.2	52.4%	.89 & .89	-	¢.6
		Simple	44.7% (+2.9%)	.83 & .87	1.3	¢.8	57.7% (+5.3%)	.89 & .89	3.0	¢2.3
		CQs	65.9% (+24.1%)	.91 & .93	13.4	¢2.5	82.7% (+30.3%)	.95 & .96	24.9	¢6.6
		CQs & ES	54.8% (+13.0%)	.88 & .90	13.0	¢1.8	75.0% (+22.6%)	.94 & .95	18.6	¢5.9
	BIRD	Baseline	28.4%	.87 & .84	-	¢.4	34.9%	.89 & .88	-	¢1.2
		Simple	30.8% (+2.4%)	.88 & .86	0.9	¢1.7	38.3% (+3.4%)	.89 & .89	1.4	¢5.3
		CQs	46.3% (+17.9%)	.92 & .90	7.0	¢4.6	66.4% (+31.5%)	.94 & .93	16.0	¢13.5
		CQs & ES	34.3% (+8.9%)	.90 & .87	5.2	¢3.0	53.3% (+18.4%)	.93 & .92	14.0	¢10.9

Table 6: The evaluation results on BIRD (after all interactions) after including the provided hints for each question.

Source	Few Shot	Method	GPT-3.5 Turbo	GPT-4 Turbo
BIRD (with hints)	0-shot	Baseline	45.8%	49.7%
		Simple	48.2% (+2.4%)	52.5% (+2.8%)
		CQs	54.2% (+8.4%)	67.1% (+17.4%)
		CQs & ES	51.2% (+5.4%)	63.4% (+13.7%)
	3-Shot	Baseline	43.8%	51.6%
		Simple	47.0% (+3.2%)	53.3% (+1.7%)
		CQs	56.0% (+12.2%)	68.1% (+16.5%)
		CQs & ES	51.4% (+7.6%)	63.9% (+12.3%)

little to no benefit but CQs and CQs & ES experience meaningful improvements.

7.4.4 Evaluation on BIRD with hints. So far, all results on BIRD benchmark do not make use of the extra hints available for each question. Table 6 presents the result for zero-shot and three-shot evaluation of our framework on the BIRD benchmark when hints are also included in the prompts. CQs provides the best accuracy in both settings and across both models. With GPT-4 Turbo, CQs is able to achieve an accuracy of 68.1% (with at most four interaction) on the development set, which is competitive with several top solutions on the BIRD leaderboard. Due to lack of space, metrics on the efficiency and cost have been deferred to the full paper [1].

7.5 Sphinteract User Study

To study the effectiveness and usability of our system, we recruited 11 participants across different roles (data scientists, engineers, and graduate students). All participants had a background in computer science and were familiar with SQL. In the user study, the participants were first briefed about NL2SQL and our system were given a browser-based implementation of the system that uses GPT-4 Turbo. The tasks in the study were divided into two parts that were performed sequentially. In the first part of the study, the participants were given 10 challenging NL questions from the BIRD and Kaggle benchmarks, along with the relevant database schema. These questions required 3 to 4 iterations in our experiments. Further, we ensured that the questions cover all types of ambiguity and a majority of the questions contain more than one type of ambiguity. Users were instructed to study the database schema and description. The objective was to interact with Sphinteract until the generated

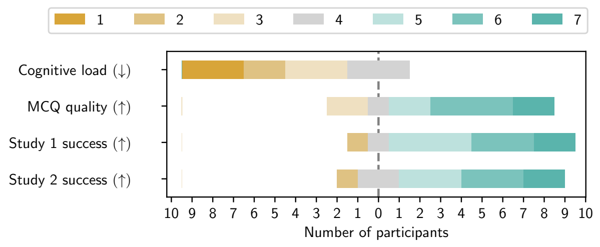


Figure 5: Feedback from Likert-scale questionnaire. Arrows indicate whether lower or higher values are better.

SQL matches their own expectation. In the second part of the study, the users looked at the same set of NL questions as the first part but were also given the benchmark ground truth queries. The objective was to interact with the clarification questions to *steer* the system towards generating the ground truth query or a query equivalent to it. Finally, participants were also asked to fill a 7-point Likert-scale [35] based questionnaire to rate their effectiveness in solving the task and system usability, quality and perceived cognitive load of the clarification questions, and provide any general feedback. The entire user study lasted between 1-2 hours. The authors took notes to record feedback after the study.

Quantitative results. The quantitative results of our user study are shown in Figure 5. As the results indicate, most of the participants felt satisfied with SPHINTERACT. They agreed that the mental load of interacting with clarifications questions was low and the overall quality of the questions generated by the system was good. The majority of participants perceived themselves to be successful on the two tasks. For the second study, upon analysis of the logs, we observed that $\sim$ 90% of the questions were solved correctly[5]. Across both the studies, the users chose the free-form feedback option in the clarification questions $\sim$ 45% of the time, demonstrating that both open and closed ended feedback are valuable to users. The average number of interactions per question was found to be 2.18. These results indicate that SPHINTERACT was successful in achieving its goal of helping users generate higher quality SQL via simple CQs while having low cognitive overhead.

Qualitative results. To derive qualitative results, we read the free-form feedback provided by the participants in the questionnaire to identify the strengths and limitations of the system. One participant commented that *"The interactive system is very friendly to work with and it can help rectify errors and ambiguities through iterations. Overall it is a very helpful tool for non-experts of SQL."*. Another participant observed that they were *"very surprised by the accuracy and efficiency of this system"*. A third participant observed that *"except one question, clarifications questions always helped. In some cases, even though the first query was decent, looking at the questions made me think about a better SQL query that would answer the question"*. The participants also identified a few opportunities for improvement in the system as well. Two participants said that they preferred interacting with the system via free-form feedback option compared to using radio buttons since the radio button option text was not exactly what they wanted. A different participant noted that they wanted the system to ask more clarification questions to address ambiguity in data values (i.e. AMBVALUE) since that was the biggest cause of ambiguity according to them. It was also noted that sometimes, the system asks questions that are quite similar to questions that have been answered before and occasionally can go into a loop of repeating a set of questions that have been asked previously.

[5]Some users had left the final query with minor variations (e.g., select an extra column).

7.5.1 Limitations and Future Work. Our research is not without limitation. In this section, we identify future work based on the feedback from the user study. First, the User Experience (UX) of the interactions requires further extension and study. For example, a better ergonomic interface for asking which columns to include in the output could be via a small widget that may support drag and drop of columns. Customization of the questions based on user preference is also an important aspect that was highlighted by the participants. This could be done via a learning module that learns from the responses of the users to the CQs. Similar (or repeated) questions can be mitigated by making use of LLMs multiple generation capability in a single call by varying the temperature setting of the LLM and discarding clarification questions with high similarity to questions already asked. In terms of the study itself, there are three aspects to improve the evaluation. First, it is important to do a large scale study with a more diverse set of participants, both for ambiguity detection to uncover more cases of how users think about ambiguity, and to understand how users prefer to engage with a real-world system. Second, the study needs to be conducted on production datasets as they can be significantly more challenging. Finally, it would be interesting to conduct a longitudinal study where the system is used on a daily basis for a longer period. In this work, we integrate user feedback into the NL2SQL pipeline to address ambiguities. However, this is not the sole approach. Contextualizing prompts through query logs, semantic models, learning from (NL, SQL) pairs, and employing fine-tuned models are alternative methods that warrant further exploration. Several practical challenges must be overcome for these approaches to be effective. For example, training or fine-tuning a model can be resource-intensive [18] and presents privacy and security risks related to exposing customer data [49, 60, 70, 71]. Additionally, incorporating models with distinct architectures can complicate deployment compared to using a universal foundation model. Specialized models can also be difficult to maintain as workloads or customer requirements change. Addressing these issues presents exciting research directions.

8 CONCLUSION

In this paper, we proposed SPHINTERACT, a framework that capitalizes on the capabilities of LLMs by incorporating user feedback directly into the SQL generation process to help LLMs in understanding a user's question. We presented different algorithms to capture various types of user feedback, utilizing either a fixed interaction budget or a self-determined stopping point. Our framework not only boosts the model's capability to generate accurate and context-sensitive SQL responses but also dynamically adapts to each user's unique requirements. Our evaluation achieves an execution accuracy of 82.7% and 66.7% on the cross-domain KaggleDBQA and BIRD benchmarks respectively, and confirmed that our framework is effective through a user study.

REFERENCES

[1] [n.d.]. sphinteract_full.pdf. https://github.com/ZhaoFuheng/Sphinteract

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

[3] Gati V Aher, Rosa I Arriaga, and Adam Tauman Kalai. 2023. Using large language models to simulate multiple humans and replicate human subject studies. In *International Conference on Machine Learning*. PMLR, 337–371.

[4] Dean Allemang and Juan Sequeda. 2024. Increasing the LLM Accuracy for Question Answering: Ontologies to the Rescue! *arXiv preprint arXiv:2405.11706* (2024).

[5] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big?. In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*. 610–623.

[6] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 17682–17690.

[7] James Brand, Ayelet Israeli, and Donald Ngwe. 2023. Using gpt for market research. *Available at SSRN 4395751* (2023).

[8] Christoph Brandt, Nadja Geisler, and Carsten Binnig. 2020. Towards robust and transparent natural language interfaces for databases. (2020).

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[10] Xinyun Chen, Maxwell Lin, Nathanael Schaerli, and Denny Zhou. 2023. Teaching Large Language Models to Self-Debug. (2023).

[11] Zui CHen, Lei Cao, Sam Madden, Ju Fan, Nan Tang, Zihui Gu, Zeyuan Shang, Chunwei Liu, Michael Cafarella, and Tim Kraska. 2023. Seed: Simple, efficient, and effective data management via large language models. *arXiv preprint arXiv:2310.00749* (2023).

[12] Bonaventure FP Dossou, Atnafu Lambebo Tonja, Oreen Yousuf, Salomey Osei, Abigail Oppong, Iyanuoluwa Shode, Oluwabusayo Olufunke Awoyomi, and Chris Emezue. 2022. AfroLM: A Self-Active Learning-based Multilingual Pretrained Language Model for 23 African Languages. (2022), 52–64.

[13] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).

[14] Ahmed Elgohary, Christopher Meek, Matthew Richardson, Adam Fourney, Gonzalo Ramos, and Ahmed Hassan. 2021. NL-EDIT: Correcting Semantic Parse Errors through Natural Language Interaction. (2021), 5599–5610.

[15] Raul Castro Fernandez, Aaron J Elmore, Michael J Franklin, Sanjay Krishnan, and Chenhao Tan. 2023. How large language models will disrupt data management. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3302–3309.

[16] Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, Alex Van Grootel, Brandon Chow, Kai Deng, Katherine Lin, Marcos Campos, K. Venkatesh Emani, Vivek Pandit, Victor Shnayder, Wenjing Wang, and Carlo Curino. 2024. NL2SQL is a solved problem... Not!. In *Conference on Innovative Data Systems Research*. https://api.semanticscholar.org/CorpusID:266729311

[17] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proceedings of the VLDB Endowment* 17, 5 (2024), 1132–1145.

[18] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proceedings of the VLDB Endowment* 17, 5 (2024), 1132–1145.

[19] Orest Gkini, Theofilos Belmpas, Georgia Koutrika, and Yannis Ioannidis. 2021. An in-depth benchmarking of text-to-sql systems. In *Proceedings of the 2021 International Conference on Management of Data*. 632–644.

[20] Barbara Grosz. 1983. Team: A transportable natural language interface system. In *Proceedings of the Conference on Applied Natural Language Processing (1983)*. Association for Computational Linguistics.

[21] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. [n.d.]. Large Language Models Cannot Self-Correct Reasoning Yet. ([n. d.]).

[22] Zezhou Huang, Pavan Kalyan Damalapati, and Eugene Wu. [n.d.]. Data Ambiguity Strikes Back: How Documentation Improves GPT's Text-to-SQL. In *NeurIPS 2023 Second Table Representation Learning Workshop*.

[23] Binyuan Hui, Xiang Shi, Ruiying Geng, Binhua Li, Yongbin Li, Jian Sun, and Xiaodan Zhu. 2021. Improving text-to-sql with schema dependency learning. *arXiv preprint arXiv:2103.04399* (2021).

[24] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. (2017), 963–973.

[25] Graham Kalton and Howard Schuman. 1982. The effect of the question on survey responses: A review. *Journal of the Royal Statistical Society Series A: Statistics in Society* 145, 1 (1982), 42–57.

[26] George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *The VLDB Journal* 32, 4 (2023), 905–936.

[27] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. 1, 2 (2019).

[28] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *Proceedings of the VLDB Endowment* 13, 10 (2020), 1737–1750.

[29] Gokhan Kul, Duc Thanh Anh Luong, Ting Xie, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. 2018. Similarity metrics for SQL query clustering. *IEEE Transactions on Knowledge and Data Engineering* 30, 12 (2018), 2408–2420.

[30] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic Evaluation of Text-to-SQL Parsers. (2021), 2261–2273.

[31] Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. 2022. Ehrsql: A practical text-to-sql benchmark for electronic health records. *Advances in Neural Information Processing Systems* 35 (2022), 15589–15601.

[32] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? *arXiv preprint arXiv:2406.01265* (2024).

[33] Fei Li and Hosagrahar V Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment* 8, 1 (2014), 73–84.

[34] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems* 36 (2024).

[35] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of Psychology* (1932).

[36] Samuel Madden, Michael Cafarella, Michael Franklin, and Tim Kraska. 2024. Databases Unbound: Querying All of the World's Bytes with AI. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4546–4554.

[37] OpenAI. 2024. OpenAI Platform. https://platform.openai.com/examples/default-sql-translate. Accessed: 30 April 2024.

[38] openai.com. 2024. GPT-4 Turbo. Retrieved October 31, 2024 from https://platform.openai.com/docs/models#gpt-4-turbo-and-gpt-4

[39] openai.com. 2024. GPT-4o. Retrieved October 31, 2024 from https://platform.openai.com/docs/models/gpt-4o

[40] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–22.

[41] 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*. 141–147.

[42] Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems* 36 (2024).

[43] Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic Prompt Optimization with "Gradient Descent" and Beam Search. (2023), 7957–7968.

[44] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[45] Joshua Robinson, Christopher Michael Rytting, and David Wingate. 2023. Leveraging Large Language Models for Multiple Choice Question Answering. arXiv:2210.12353 [cs.CL]

[46] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. 2024. Mathematical discoveries from program search with large language models. *Nature* 625, 7995 (2024), 468–475.

[47] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R Mittal, and Fatma Özcan. 2016. ATHENA: an ontology-driven system for natural language querying over relational data stores. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1209–1220.

[48] Juan Sequeda, Dean Allemang, and Bryon Jacob. 2024. A benchmark to understand the role of knowledge graphs on large language model's accuracy for question answering on enterprise SQL databases. In *Proceedings of the 7th Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and*

Network Data Analytics (NDA). 1–12.

[49] Liang Shi, Zhengju Tang, and Zhi Yang. 2024. A Survey on Employing Large Language Models for Text-to-SQL Tasks. *arXiv preprint arXiv:2407.15186* (2024).

[50] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755* (2024).

[51] Yuan Tian, Zheng Zhang, Zheng Ning, Toby Li, Jonathan K Kummerfeld, and Tianyi Zhang. 2023. Interactive text-to-SQL generation via editable step-by-step explanations. (2023), 16149–16166.

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

[53] 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. (2020), 7567–7578.

[54] Lihan Wang, Bowen Qin, Binyuan Hui, Bowen Li, Min Yang, Bailin Wang, Binhua Li, Jian Sun, Fei Huang, Luo Si, et al. 2022. Proton: Probing schema linking information from pre-trained language models for text-to-sql parsing. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1889–1898.

[55] Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *arXiv preprint arXiv:2401.04398* (2024).

[56] David HD Warren and Fernando CN Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *American journal of computational linguistics* 8, 3-4 (1982), 110–122.

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

[58] Peter West, Ximing Lu, Nouha Dziri, Faeze Brahman, Linjie Li, Jena D Hwang, Liwei Jiang, Jillian Fisher, Abhilasha Ravichander, Khyathi Chandu, et al. 2023. THE GENERATIVE AI PARADOX:"What It Can Create, It May Not Understand". In *The Twelfth International Conference on Learning Representations*.

[59] Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. 2024. Self-play preference optimization for language model alignment. *arXiv preprint arXiv:2405.00675* (2024).

[60] Siqiao Xue, Caigao Jiang, Wenhui Shi, Fangyin Cheng, Keting Chen, Hongjun Yang, Zhiping Zhang, Jianshan He, Hongyang Zhang, Ganglin Wei, et al. 2023. Db-gpt: Empowering database interactions with private large language models. *arXiv preprint arXiv:2312.17449* (2023).

[61] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–26.

[62] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems* 36 (2024).

[63] 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. (2019), 5447–5458.

[64] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. (2018), 3911–3921.

[65] Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, et al. 2019. SPARC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

[66] John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*. 1050–1055.

[67] Ningyu Zhang, Luoqiu Li, Xiang Chen, Shumin Deng, Zhen Bi, Chuanqi Tan, Fei Huang, and Huajun Chen. 2021. Differentiable prompt makes pre-trained language models better few-shot learners. *arXiv preprint arXiv:2108.13161* (2021).

[68] Xinlu Zhang, Yujie Lu, Weizhi Wang, An Yan, Jun Yan, Lianke Qin, Heng Wang, Xifeng Yan, William Yang Wang, and Linda Ruth Petzold. 2023. Gpt-4v (ision) as a generalist evaluator for vision-language tasks. *arXiv preprint arXiv:2311.01361* (2023).

[69] Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2023. ScienceBenchmark: A Complex Real-World Benchmark for Evaluating Natural Language to SQL Systems. *Proceedings of the VLDB Endowment* 17, 4 (2023), 685–698.

[70] Yunjia Zhang, Avrilia Floratou, Joyce Cahoon, Subru Krishnan, Andreas C Müller, Dalitso Banda, Fotis Psallidas, and Jignesh M Patel. 2023. Schema matching using pre-trained language models. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1558–1571.

[71] Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M Patel. [n.d.]. ReAcTable: Enhancing ReAct for Table Question Answering. ([n. d.]).

[72] Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M Patel. 2024. ReAcTable: Enhancing ReAct for Table Question Answering. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1981–1994.

[73] Fuheng Zhao, Divyakant Agrawal, and Amr El Abbadi. 2024. Hybrid Querying Over Relational Databases and Large Language Models. *arXiv preprint arXiv:2408.00884* (2024).

[74] Fuheng Zhao, Lawrence Lim, Ishtiyaque Ahmad, Divyakant Agrawal, and Amr El Abbadi. 2023. LLM-SQL-Solver: Can LLMs Determine SQL Equivalence? *arXiv preprint arXiv:2312.10321* (2023).

[75] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103* (2017).

[76] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911* (2023).