

Multi-Turn Interactions for Text-to-SQL with Large Language Models

Guanming Xiong
Peking University
Beijing, China
gm_xiong@pku.edu.cn

Junwei Bao*
Zuoyebang Education Technology
Co., Ltd.
Beijing, China
baojunwei001@gmail.com

Hongfei Jiang
Zuoyebang Education Technology
Co., Ltd.
Beijing, China
jianghongfei@zuoyebang.com

Yang Song
Zuoyebang Education Technology
Co., Ltd.
Beijing, China
songyang@zuoyebang.com

Wen Zhao
Peking University
Beijing, China
zhaowen@pku.edu.cn

Abstract

This study explores text-to-SQL parsing by leveraging the powerful reasoning capabilities of large language models (LLMs). Despite recent advancements, existing LLM-based methods are still inefficient and struggle to handle cases with wide tables effectively. Furthermore, current interaction-based approaches either lack a step-by-step, interpretable SQL generation process or fail to provide a universally applicable interaction design. To address these challenges, we introduce Interactive-T2S, a framework that generates SQL queries through direct interactions with databases. This framework includes four general tools that facilitate proactive and efficient information retrieval by the LLM. Additionally, we have developed detailed exemplars to demonstrate the step-wise reasoning processes within our framework. Our approach achieves advanced performance on the Spider and BIRD datasets as well as their variants. Notably, we obtain state-of-the-art results on the BIRD leaderboard under the setting without oracle knowledge, demonstrating the effectiveness of our method.¹

CCS Concepts

- Computing methodologies → Natural language processing.
- Information systems → Question answering; Search interfaces.

Keywords

Text-to-SQL, Large Language Model, Low-resource

*Corresponding author.

¹Code and data are available at: <https://github.com/JimXiongGM/Interactive-Text-to-SQL>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CIKM '25, Seoul, Republic of Korea

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 979-8-4007-2040-6/2025/11
<https://doi.org/10.1145/3746252.3761052>

ACM Reference Format:

Guanming Xiong, Junwei Bao, Hongfei Jiang, Yang Song, and Wen Zhao. 2025. Multi-Turn Interactions for Text-to-SQL with Large Language Models. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management (CIKM '25), November 10–14, 2025, Seoul, Republic of Korea*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3746252.3761052>

1 Introduction

Text-to-SQL technology, which translates natural language questions into executable SQL queries, has emerged as a crucial field of research. This technology empowers non-experts to interact with relational databases (DBs), which have become ubiquitous in the era of big data [11]. A significant challenge in this field is designing a text-to-SQL system that operates accurately and efficiently within resource constraints.

The emergence of large language models (LLMs), such as ChatGPT [23] and GPT-4 [22], has opened new avenues for enhancing text-to-SQL systems. These models have shown promising capabilities in reasoning and few-shot learning, establishing new benchmarks in this domain [6, 24].

Recent advancements in text-to-SQL research encompass two primary perspectives: prompt optimization and interaction strategies [11]. Prompt optimization focuses on crafting prompts that guide LLMs to generate accurate SQL queries. This involves constructing precise schema linking, leveraging similar examples, and employing effective question decomposition methods [6, 37]. Interaction strategies, on the other hand, center around designing methods to refine SQL queries through execution-based feedback [1, 25, 28]. Recent approaches also introduce interactive models that leverage specific tools to interact with DBs, yielding significant improvements [7, 12].

Despite these advancements, text-to-SQL systems face several pressing challenges.

Inefficiency when scaling to wide tables. For schema linking, existing LLM-based methods typically input all columns from a table (or all tables), consuming substantial LLM window sizes and struggling to scale efficiently. Additionally, these approaches incur increasing costs with the growth in the number of columns and

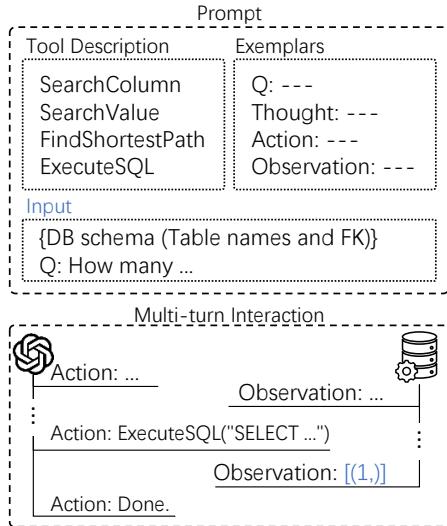


Figure 1: Overview of the interactive process.

generally lack support for locating cell values, a critical feature given the frequent updates in real-world DBs.

Design deficiencies in interaction-based methods. Current methodologies based on execution refinement directly generate complete final SQL queries, which are coarse-grained and lack a step-by-step interpretable process for SQL generation. Furthermore, although multi-interaction-based methods hold promise, they exhibit significant shortcomings. For instance, [12] does not provide tools for searching cell values, reducing its usability. [7] encapsulates six SQL functions into separate functional tools, which adds unnecessary complexity when SQL execution alone would suffice. Additionally, these methods do not adequately address scalability concerns.

Resource scarcity for annotating text-SQL data. Current works emphasize prompt optimization by dynamically selecting exemplars based on similarity metrics. However, this approach assumes the availability of extensive training data as a candidate pool for exemplars while being unrealistic to expect user queries to consistently align with the training data distribution. Moreover, these techniques require large annotated datasets that are costly to create, making it crucial to explore methods for low-resource settings.

Inspired by [34], we propose the Interactive-Text-to-SQL framework, which leverages the reasoning capabilities of LLMs to interact with DBs through a step-by-step, conversational process. As depicted in Figure 1, this framework conceptualizes the LLM as an agent and DBs as the environment, operating under a thought-action paradigm. Specifically, the LLM is required to think and then act, interacting with the DBs through a specially designed toolkit. We provided only two annotated exemplars with complete interactive processes as demonstrations for in-context learning, prompting the LLM to complete the task. Comprehensive experiments on the Spider-Dev, BIRD-Dev, and their variant datasets showcased that our approach achieves significant results with minimal exemplar input.

Our primary contributions are summarized as follows:

- Propose the Interactive-T2S, a novel framework for generating SQL queries through multi-turn interactions with DBs using LLMs.
- Design a unified interaction logic with four general tools that can effectively handle wide tables.
- Proof through extensive experiments that our method performs exceptionally well with just two exemplars.

2 Related Work

Recently, large language models (LLMs) have demonstrated remarkable reasoning capabilities [22, 23], offering new opportunities for text-to-SQL systems. Current LLM-based methods can generally be categorized into two aspects: prompt optimization-based and interaction-based.

Prompt optimization-based approaches enhance SQL query generation by optimizing prompts for LLMs through schema linking and exemplar selection. **Schema linking** aligns natural language questions with DB schema elements. Methods like [24] input entire DB schema and questions into the model to identify relevant tables and columns. Hierarchical classification approaches by [13] first select relevant tables, then pinpoint columns. [31] show focusing on tables alone can outperform methods targeting columns. However, these models struggle with scalability due to LLMs' limited window sizes. **Exemplar selection** involves choosing similar questions and queries to guide the model. Techniques by [9] and [20] utilize structural and syntactic similarities. [6, 19] prioritize candidates based on question and SQL similarities, while [37] uses a hybrid approach. High-diversity demonstrations are explored by [32] to improve retrieval systems. However, these methods often assume complete training dataset access, limiting practical applicability.

Interaction-based approaches like [35] guide LLMs to interact with the environment to accomplish tasks. Some works focus on refining SQL based on **execution results**. [28] introduced a framework that executes sampled SQL queries to select the most accurate translation based on execution risk. [21] incorporated a learned verifier estimating query correctness. [1] developed a method for LLMs to autonomously debug and refine SQL by examining execution outcomes. [8] applied a dynamic revision chain that uses execution feedback for correcting SQL semantic discrepancies. Similar to our work, [12] treats the **LLM as agent** and designs tools for extracting relevant information to interact with DBs. [7] designs navigational and functional tools, but their tool design and interaction logic are suboptimal.

3 Approach

3.1 Problem Formulation

This study investigates the text-to-SQL task. A relational database (DB) is formally represented as $\mathcal{D} = \{\mathcal{T}, \mathcal{C}, \mathcal{V}\}$, where $\mathcal{T} = \{t_1, t_2, \dots, t_{|\mathcal{T}|}\}$ is a set of tables, $\mathcal{C} = \{c_1, c_2, \dots, c_{|\mathcal{C}|}\}$ is a set of columns, and $\mathcal{V} = \{v_1, v_2, \dots, v_{|\mathcal{V}|}\}$ is a set of cell values. Each table t_i comprises a set of columns c_i and each column c_i encompasses a set of cell values v_i .

²Due to limited space, some thought processes were omitted.

Q: Which male professors participated in the soccer activity? List their names.



Thought: I need to: 1. Find the cell value related to the 'Soccer activity', and 2. Identify the columns 'male professor' and 'professor name'.
Action: `SearchValue("Soccer activity")`

Observation: `[{"contents": "Soccer", "table": "Activity", "column": "name"}]`



Action: `SearchColumn(["male professor", "professor name"])`

Observation: `{"male professor": [{"column": "Sex", "table": "Faculty", "statistics": "categorical field. [M', 'F']"}, {"column": "Sex", "table": "Student", "statistics": "categorical field. [F', 'M']"}, {"column": "Rank", "table": "Faculty", "statistics": "categorical field. [Instructor', 'Professor', 'AsstProf"]}], "professor name": [{"column": "Lname", "table": "Faculty", "statistics": "text filed. e.g. Kumar, Goldman, ..."}, {"column": "Fname", "table": "Faculty", "statistics": "text filed. e.g. Baruch, Paul, Brian, ..."}]}`



Thought: To summarize, the columns that need to be selected are `Faculty.Fname`, `Faculty.Lname`, and the column that need to be filtered is `Activity.name`.
Action: `FindShortestPath(start="Faculty.Fname", end="Activity.name")`

Observation: `Faculty.Fname <-> Faculty_Participates_in.FacID = Faculty.FacID <-> Faculty_Participates_in.actid = Activity.actid <-> Activity.activity_name`



Action: `ExecuteSQL("SELECT Faculty.Fname, Faculty.Lname FROM Faculty JOIN Faculty_Participates_in ON Faculty.FacID = Faculty_Participates_in.FacID JOIN Activity ON Faculty_Participates_in.actid = Activity.actid WHERE Activity.name = 'Soccer' AND Faculty.Sex = 'M' AND Faculty.Rank = 'Professor'")`

Observation: `[("Michael", "Goodrich")]`



Action: Done.

Figure 2: An example of the interactive process.²

Following [16], we further define foreign key relations $\mathcal{R} = \{(c_k^i, c_h^j) | c_k^i, c_h^j \in C\}$, where each pair (c_k^i, c_h^j) denotes a foreign key relation between column c_k^i in table i and column c_h^j in table j . The database schema $\mathcal{S} = \{\mathcal{T}, \mathcal{C}, \mathcal{R}\}$ constitutes a set of tables, columns, and foreign key relations.

Formally, given a question q and a database \mathcal{D} , the objective of the text-to-SQL task is to translate the question q into a SQL query l that can be executed on \mathcal{D} to answer the question.

3.2 Overview

Recent advancements in large language models (LLMs) have highlighted their impressive capabilities in few-shot learning and logical reasoning. Nevertheless, designing scalable solutions for interpretable and step-by-step SQL generation in low-resource scenarios remains challenging. In response, we introduce Interactive-T2S, a novel interactive method for text-to-SQL translation. This method treats the LLM as an agent interacting with a database environment, enhancing SQL generation through structured dialogic interactions. We developed a unified interaction logic with four generic tools to help LLMs identify relevant information and discern relationships across multiple tables. The example in Figure 2 illustrates this interactive process. Different colors highlight how the corresponding elements can be located.

3.3 Tools for Database

We break down the process of generating SQL into three steps: searching for relevant columns and cell values, identifying the join relationships between tables where columns reside, and refining

the prediction based on the execution. In line with this principle, we introduce the following four tools.

SearchColumn(semantic) enhances the efficiency of LLMs by identifying the most relevant columns and excluding non-essential data. It concatenates and vectorizes the names and descriptions of each column, then ranks these columns according to their similarity to the parameter `semantic`. Furthermore, following the methodology proposed by [19], we calculate and return the statistical characteristics of each column's cell values.

SearchValue(value, table=None, column=None) is designed to locate cell values across the entire DB. Similar to the fuzzy match tool described in [7], we utilize BM25 to search for cell values within the DB. If the parameters `table` or `column` are specified, the tool will conduct searches within the designated table or column.

FindShortestPath(start, end) is designed to efficiently identify the shortest path between two columns within a DB schema, based on foreign key relationships. Existing methods heavily rely on the intrinsic capabilities of LLM to perform joins across multiple tables, which becomes impractical with scenarios involving extensive joins or a large number of columns. In contrast, the number of tables requiring joins is dictated solely by the DB schema design, rather than the semantic content of the question. By modeling the DB schema as an undirected graph where columns are nodes and edges are defined by column relationships and foreign keys, this tool simplifies multi-table joins and reduces LLM workload.

ExecuteSQL(sql) provides the capability to execute SQL queries directly, offering significant flexibility.

3.4 Interactive Process

Given a question q , we first construct a prompt text:

$$\text{Prompt} = \{\text{Inst}, E, S_q, q\} \quad (1)$$

where Inst denotes the pre-written instruction text, which encompasses descriptions of tools, usage guidelines, and the required format. $E = [(S_0, e_0, \dots, e_n), \dots]$ represents a list of demonstrations, each consisting of a database schema S_i and n exemplars e with full interactive process.

In each turn T , we prompt the LLM generate an action based on the Prompt and the historical interaction H . Specifically, this process is described by:

$$a_T = \text{LLM}(\{\text{Prompt}, H\}) \quad (2)$$

$$H = \{c_0, a_0, o_0, \dots, c_{T-1}, a_{T-1}, o_{T-1}\} \quad (3)$$

where c denotes the intermediate thought process, an action a belongs to the set {SearchColumn, SearchValue, FindShortestPath, ExecuteSQL, Done}, and the observation o results from executing an action, defined as $o_T = \text{Tool}(a_T)$.

3.5 General Solution for Text-to-SQL

We propose a general and unified interaction logic for generating SQL queries, as the example illustrated in Figure 2. The process begins with **locating elements**. Initially, the LLM is tasked with decomposing the user’s question into a conceptual plan (c_0), which is flexibly designed to adapt to the semantics of the question, enhancing comprehension for both LLMs and humans. Following this, the agent is required to generate a thought c and an action a_T aimed at identifying pertinent columns and cell values within the DB. The next phase is **joining tables**. As the question shown in Figure 2, the selected columns are `Faculty.Fname` and `Faculty.Lname`, while the filtered column is `Activity.name` (with the constraint “activity = soccer”). Note that for the tool, start and end are interchangeable since the schema is treated as an undirected graph; the distinction between selected and filtered columns is only for organizing the thought process. The final phase is **execute SQL**, where the LLM executes the constructed SQL query to retrieve the desired results. This query execution is deemed the final output.

4 Experiment

4.1 Dataset

Table 1: Statistics of the datasets.

Dataset	#Item		#DB		Statistics	
	Train	Dev	Train	Dev	CVR	CVCR
Spider	8,659	1,034	146	20	10.25	87.14
- DK	-	535	-	10	11.59	40.79
- Syn	7,000	1,034	140	20	10.44	76.06
- Realistic	-	508	-	19	16.54	84.96
BIRD	9,428	1,534	69	11	68.38	63.68
FinC (Original)					68.87	33.66
- (SQLC)	-	106	-	1	70.75	32.04
- (DataC)					70.75	34.95
Mini-Dev	-	500	-	11	70.60	59.63

Spider [36] is a widely used cross-domain text-to-SQL benchmark. We utilize the development set. Building on this, **Spider-DK** [5] challenges parsers with domain knowledge reasoning and real-world question paraphrases to evaluate cross-domain generalization. **Spider-Syn** [4] introduces synonyms for schema tokens to prevent reliance on string matching. **Spider-Realistic** [2] further challenges models by revising questions to omit explicit column names, thus testing text-table alignment capabilities.

BIRD [18] is notable for its complexity, featuring intricate instructions over highly complex databases. BIRD has two settings: with and without external knowledge evidence (oracle knowledge), which provides specific information needed for answering questions. We report results on both the development and test sets in our experiments. **BIRD-Financial Corrected (FinC)** [33] addresses the issue of noise in the BIRD benchmark, particularly the uneven distribution of incorrect SQL queries that affects its reliability. They have revised the dataset under the financial domain, which includes 106 question and SQL query pairs, representing approximately 7.5% of the development data. The correction includes SQL-only corrections (**SQLC**) and corrections for both SQL queries and noisy questions (**DataC**). **BIRD Mini-Dev** dataset offers 500 high-quality text-SQL pairs derived from community feedback. We utilize the SQLite version. Following [7], we argue that using oracle knowledge is unreasonable; however, we still report the results of both for comparison.

The detailed statistics are listed in Table 1. We also provide statistics for the latest Spider 2.0 dataset in Appendix 6.4.

4.2 Baselines

To comprehensively evaluate our approach, we have selected various state-of-the-art (SOTA) baseline models.

Fine-tuning (FT) on full data. CodeS [17], a series of pre-trained language models, addresses schema linking and domain adaptation through incremental pre-training on a SQL-centric corpus, strategic prompts, and bi-directional data augmentation. We select the best results in the Supervised Fine-Tuning (SFT) setting.

Prompt-based methods. We selected recent prompt-based methods using GPT-4 models. In the context of in-context learning [3], these can be categorized into **Selection-based** and **Fixed Few-shot**: the former dynamically selects exemplars from training data based on similarity, while the latter uses a fixed set of examples.

Notably, few works have addressed the BIRD dataset series without oracle knowledge. Therefore, we reimplemented several baselines following [33].

4.3 Implementation Details

We employ OpenAI’s gpt-4o-2024-05-13 as the LLM agent in our experiments. To prevent excessive token consumption, we set a maximum of 12 interaction rounds and restrict the length of each observation’s returned results. The few-shot exemplars are manually constructed. Specifically, we randomly selected and analyzed several challenging cases, annotating two representative examples each for the Spider and BIRD dataset series. Details regarding the design of prompt texts and the implementation of tools can be found in Appendix 6.1 and Appendix 6.2.

4.4 Evaluation Metrics

Following [25] and [27], we report exact match accuracy (EM) [36] and execution accuracy (EX) [38] for the Spider dataset series. EM requires each component of the predicted SQL to match the gold SQL, but it cannot handle cases where multiple correct answers exist [30, 37], so we report it only for reference. EX, which requires the execution result of the predicted SQL to be correct, is generally more precise. For the BIRD-Dev and FinC datasets, we use the EX metric. For the Mini-Dev dataset, we also report the Soft F1-Score [18].³

4.5 Main Results

Analysis of Spider-Dev and its variants. Experimental results are presented in Table 2. Our method shows competitive performance across the Spider-DK, Syn, and Realistic datasets. Selection-based methods, which choose similar exemplars from training data, outperform fixed few-shot approaches, as confirmed by the EM metric. However, the performance gap narrows significantly across the three variant datasets, suggesting selection-based methods assume similar data distributions between development and training sets, indicating their inadequate generalization capabilities.

In fixed few-shot methods, both TA-SQL and SL+CC+RS perform well because they utilize the entire DB schema (including all columns) as prompt text for schema linking, which is feasible due to the relatively small size of the DB. Table 6 highlights that Spider-Dev has only an average of 307 schema tokens per DB. Despite the complexity of TA-SQL, which uses intricate modules for generating SQL and pandas-like APIs for reasoning, our design employs a simple yet effective unified interaction logic. SL+CC+RS performs worse when incorporating domain knowledge (DK dataset), highlighting our approach’s superior generalization.

We contend that our method’s performance on Spider-Dev is limited by the dataset’s simplicity and ambiguity, as elaborated in subsequent sections on difficulty analysis and error analysis, respectively. We also conducted experiments on the Spider 2.0 dataset, as detailed in Appendix 6.4.

Analysis of BIRD-Dev and its variants. In our analysis, following the critiques by [7] regarding the impracticality of oracle knowledge in real-world applications, we focus primarily on settings without oracle knowledge. Our method surpasses the SOTA on the BIRD-Dev dataset by 2.87%, as shown in Table 3.⁹ Furthermore, we reimplemented the Zero-shot and DIN-SQL on both Mini-Dev and BIRD-FinC dataset, as conducted by [33]. The results are provided in Table 5. Across all datasets, our method sets a new standard in the setting without oracle knowledge, establishing new SOTA results.

³We omit the valid efficiency score (VES) and the Reward-based Valid Efficiency Score (R-VES) as these metrics depend on hardware performance.

⁴For Spider-Dev, we reprinted GPT-4+Graphix-T5. For others, we chose GPT-4+FastRAText.

⁵The task-aligned logical synthesis module uses 7-shot.

⁶Results tagged with # are reprinted from the official leaderboards: <https://BIRD-bench.github.io> and https://github.com/BIRD-bench/mini_dev.

⁷Results tagged with † indicates we reimplemented the results.

⁸This paper uses the OpenAI GPT-4o tokenizer by default.

⁹Our leaderboard score is 54.11.

4.6 The Difficulty Analysis of Locating Cell Values

The difficulty of SQL generation is often superficially measured by the presence of specific SQL keywords, as in prior work [18, 36]. In contrast, we propose a more rigorous and fine-grained assessment based on the challenges of accurately locating schema elements, identifying cell values, and performing table joins. In this section, we focus on the problem of cell value localization, while the other aspects are discussed in subsequent sections.

Inspired by the Mention Cover Rate metric from knowledge-based QA [34], we introduce two metrics: Cell Value Rate (CVR) and Cell Value Cover Rate (CVCR). CVR measures the proportion of SQL queries with value constraints, while CVCR indicates how often these constraint values appear directly in the questions. These metrics help quantify the difficulty of locating cell values.

Statistics presented in Table 1 reveal significant disparities between the Spider and BIRD dataset series. Specifically, the BIRD series requires the identification of cell values in approximately seven times as many cases as the Spider series. Notably, in Spider-Dev, around 87% of cases requiring cell value identification include the golden cell value directly in the question, simplifying the SQL generation process. Conversely, Spider-DK reduces the CVCR to 40.79%, substantially increasing the complexity and leading to a notable performance drop in SL+CC+RS. These findings underscore the necessity of developing tools like the SearchValue tool in our framework, which aids LLMs in pinpointing cell values.

4.7 The Efficiency Analysis of Schema Linking

In this section, we analyze the efficiency of schema linking using DIN-SQL as a case study. Table 7 lists the number of fixed prompt text (input) tokens across the four modules of DIN-SQL, as well as the estimated total prompt tokens per case based on the average tokens per DB detailed in Table 6. Given that DIN-SQL utilizes the complete DB schema in each module, the average prompt tokens per case are calculated as the sum of total fixed tokens and four times the average tokens per DB, resulting in about 12.8k and 21.6k for Spider-Dev and BIRD-Dev, respectively.

Comparatively, our method, as presented in Table 8, requires only 4.6k and 4.7k tokens per case, which corresponds to approximately 36% and 22% of the tokens required by DIN-SQL, respectively. This efficiency stems from our method’s dynamic retrieval of necessary information, which is not affected by the length of the DB schema, demonstrating our method’s scalability. It is important to note that decoder-only LLMs employ causal decoding with KV-Cache techniques [29], which enables computational efficiency when processing repetitive prefixes. Our interactive approach leveraging the OpenAI API benefits from these optimizations in terms of reduced computational costs.

4.8 Ablation Study

In this section, we explore the impact of the FindShortestPath tool. We categorized and sampled cases from Spider-Dev and BIRD-Dev based on the number of table joins (2, 3, 4+), creating two subsets, each containing 50 cases selected randomly or supplemented from variant datasets if necessary. As depicted in Table 10, the tool significantly enhanced performance in scenarios requiring joins across

	Method	Spider-Dev		Spider-DK		Spider-Syn		Spider-Realistic	
		EM	EX	EM	EX	EM	EX	EM	EX
FT SOTA	CodeS (SFT) [17]	-	85.4	-	70.7	-	77.0	-	83.1
Selection-based	DAIL-SQL [6]	71.9	83.6	-	-	-	-	-	76.0
	AST Norm ⁴ [27]	77.3	86.6	59.1	72.3	61.3	74.4	66.1	80.9
	PURPLE [26]	80.5	87.8	61.7	75.3	63.3	74.0	71.1	79.9
Fixed Few-shot	TA-SQL (7-shot) ⁵ [25]	44.3	85.0	-	72.9	-	-	-	79.5
	SL+CC+RS (6-shot) [31]	-	86.2	-	67.2	-	78.1	-	82.8
	Ours (2-shot)	52.0	84.8	41.5	75.5	44.9	78.7	55.7	81.3

Table 2: Results on Spider-Dev and its variants.

Table 3: Results on BIRD-Dev.⁶

Model		EX
<i>w/ Oracle Knowledge</i>		
FT SOTA	CodeS (SFT)	58.47
SOTA	Distillery + GPT-4o #	67.21
<i>w/o Oracle Knowledge</i>		
FT SOTA	CodeS (SFT)	47.91
SOTA	ExSL + granite-20b-code #	51.69
Fixed Few-shot	StructGPT [12]	31.80
	FUXI [7]	42.90
	Ours	54.56

Table 4: Results on BIRD-FinC. The EX metric is reported.⁷

Model	Original	SQLC	DataC
<i>w/ Oracle Knowledge</i>			
Zero-shot (GPT-4)	38.09	48.11	55.66
Zero-shot (GPT-4o) †	50.00	57.55	62.26
DIN-SQL (GPT-3.5)	34.91	38.68	47.16
DIN-SQL (GPT-4o) †	43.40	50.94	65.09
Ours	54.72	65.09	69.81
<i>w/o Oracle Knowledge</i>			
Zero-shot (GPT-4o) †	26.42	31.13	35.85
DIN-SQL (GPT-4o) †	39.62	47.17	56.60
Ours	44.34	49.06	58.49

four or more tables. Additionally, we analyzed the distribution of join table counts within the golden SQL queries, as listed in Table 9. Notably, cases with four or more table joins constitute less than 2% of both datasets, underscoring the substantial potential of our

Table 5: Results on BIRD Mini-Dev.

Model	EX	Soft F1-Score
<i>w/ Oracle Knowledge</i>		
GPT-4 #	47.80	52.69
TA + GPT-4-turbo #	58.00	62.40
TA + GPT-4o #	63.00	66.97
Ours	58.80	63.07
<i>w/o Oracle Knowledge</i>		
Zero-shot (GPT-4o) †	28.00	31.99
DIN-SQL (GPT-4o) †	36.80	40.60
Ours	46.60	50.75

Table 6: Statistics of the average number of tables (Tb) per DB, columns (Col) per table, foreign keys (FK) per DB, and schema tokens (STk) per DB.⁸

Dataset (Dev)	Tb/DB	Col/Tb	FK/DB	STk/DB
Spider	4.00	5.49	3.25	307
BIRD	6.82	10.64	9.55	3,324

Table 7: Statistics of the prompt token consumption in DIN-SQL, which consists of four modules: Schema Linking (SLink), Classification & Decomposition (QCLDe), SQL Generation (SQLGen), and Self-correction (SelfC).

Tokens of Fixed Prompt Text			
SLink	QCLDe	SQLGen	SelfC
3,411	4,028	3,170	997
Total (Fixed)			
	Spider-Dev	BIRD-Dev	
11,606	12,834	21,622	

tool. Importantly, the complexity of a question’s semantics does not necessarily correlate with the number of table joins, which depend solely on the DB’s design. Hence, the FindShortestPath

Table 8: Statistics of token consumption of fixed prompt text (#Tk of Fixed), including tool descriptions (Desc) and exemplars (Exem), as well as the average number of interaction rounds (Avg. R), average total number of tokens (Avg. Tk), and average cost (Avg. \$). The term “kg” refers to oracle knowledge.

Dataset (Dev)	#Tk of Fixed		Avg. R	Avg. Tk	Avg. \$
	Desc	Exem			
Spider		2,308	4.36	4,634	0.10
BIRD (w/ kg)	1,160	1,608	5.59	4,474	0.12
BIRD (w/o kg)		1,690	5.75	4,715	0.13

Table 9: Distribution of the number of tables involved in golden SQL queries.

Dataset (Dev)	Number of Tables				Avg.
	1	2	3	4+	
Spider	60.54	30.95	6.96	1.55	1.50
BIRD	25.68	58.93	13.95	1.43	1.92

Table 10: The impact of the FindShortestPath tool. The EX metric is reported.

Dataset	Number of Tables			Avg.
	2	3	4+	
Spider	80	84	76	80.00
w/o findpath	74	80	54	69.33
Gain	6	4	22	10.67
BIRD	70	58	40	58.00
w/o findpath	62	57	28	51.33
Gain	8	1	12	6.67

tool is designed to decouple the path-finding process from direct inference by LLMs from the DB schema, thereby alleviating unnecessary reasoning burdens and ensuring the system’s scalability. We also conducted the experimental results of open-source LLMs in Appendix 6.5.

4.9 Error Analysis

In this section, we performed an error analysis by randomly sampling 100 cases from both the Spider-Dev and BIRD-Dev datasets, consistent with the original data distribution. The results are documented in Table 11. We categorized the errors into two main types: Mismatch and Error. A Mismatch refers to cases where the generated SQL does not match the golden SQL but maintains semantic consistency with the question. Conversely, an Error indicates an incorrect SQL generation.

For Mismatch types: **Golden Wrong** highlights mismatches where the golden SQL does not align with the question requirements. **Golden Empty** refers to scenarios in which the golden SQL query produces no results. Within our framework, this occurrence might lead the model to mistakenly perceive its generated query as

Table 11: Distribution of error types.

Name	Spider-Dev	BIRD-Dev
	Mismatch	
Golden Wrong	10	2
Golden Empty	35	4
Ambiguous	25	14
Distinct	6	8
Format	5	18
Lacking Info	-	20
Total	82	86

Name	Spider-Dev	BIRD-Dev
	Error	
Selected Column	6	20
Filtered Column	3	4
Reasoning	9	10
Total	18	34

incorrect during the intermediate steps. Such a misperception can lead to unnecessary interactions, ultimately resulting in erroneous outputs. **Ambiguous** represents cases where multiple valid SQL queries could correctly answer the question, but differ from the single golden reference, highlighting the limitation of having only one reference solution. **Distinct** denotes that the predicted and the golden differ by only one DISTINCT keyword, which is not explicitly required in the question. **Format** indicates correct queries that are hindered by format inconsistencies, such as NULL handling or time fields, resulting in different execution outcomes. **Lacking Info** applies exclusively to the BIRD dataset, signifying cases where missing oracle knowledge makes the question challenging to answer accurately.

For Error types: **Selected Column** points to inaccuracies in the prediction of the selected column, often due to numerous similar column names. **Filtered Column** refers to errors in predicting the correct filtered column. **Reasoning** indicates fundamental issues with the SQL structure, beyond mere column inaccuracies.

Surprisingly, only 18% and 34% of all erroneous cases are caused by reasoning errors. Therefore, we argue that different datasets possess their unique annotation styles, and it is exceedingly challenging to identify and align with these styles using very few demonstrations.

Acknowledgments

This work was supported by the National Key Research and Development Program of China under Grant No. 2023YFC3304404.

5 Conclusion

Interactive-T2S is a text-to-SQL approach that leverages a LLM as an agent to generate SQL queries through multi-round interactions with a database. We designed a unified tool and interaction methodology for schema linking, cell value localization, table joining, and query refinement based on execution results. Additionally, we employed a few-shot learning strategy to guide the LLM in incrementally generating SQL queries. Experimental results demonstrate that our method achieves SOTA results with just two exemplars.

6 Appendix

This appendix provides detailed experimental results and offers further discussion.

6.1 More Implementation Details

The prompt text is shown in Figure 3. The few-shot exemplars are manually constructed. For Spider, we observed that queries related to the “academic” database returned empty results, which could mislead the LLM into thinking it made an error and continue exploring unnecessarily. Therefore, we selected one such case as an example.

6.2 Tool Implementation Details

SearchColumn(semantic). The tool SearchColumn is designed to rank database columns based on their relevance to the semantic parameter. For the Spider dataset, column and table names are embedded using the template “a column named {column_name} in table {table_name}”. For the BIRD dataset, the template “a column named {column_name} in table {table_name} about {desc}” is used, where {desc} includes descriptions of the column provided by the dataset [18].

We utilize the OpenAI text-embedding-3-large API to generate vectors and employ Chroma¹⁰ to index and search.

The tool will return the following features for each column:

- column_name: the name of the column.
- table_name: the name of the table.
- column_type: the data type of the column.
- column_desc: the description of the column.
- column_statistics: the statistics of the cell values in the column.

In our approach, the column_name, table_name, and column_type are extracted through SQL queries. For describing the columns semantically, we adopt the “semantic name” as the “column_desc” in the context of the Spider dataset, following the methodology outlined by [16]. For the BIRD dataset, we utilize the column descriptions as provided within the original dataset. Furthermore, we enhance the representation of each column by computing statistical features from the cell values, an extension to the “cell value reference” presented by [16]. Specifically, for text-based columns, we randomly sample cell values and return the first 100 characters; for numeric or date types, we calculate and return the maximum and minimum values. This enriched feature set aids in a deeper understanding and processing of column data.

SearchValue(value). This tool is designed for searching the values within a column utilizing Elasticsearch, where only the text fields are indexed.

FindShortestPath(start, end). This tool computes the shortest path between two nodes in a graph. It leverages the NetworkX [10] library, a powerful tool for the analysis of complex networks.

ExecuteSQL(sql). This tool executes a provided SQL query using the SQLite3 library in Python 3.

6.3 System Configurations

Table 12 presents the parameters for invoking the OpenAI API.

¹⁰<https://github.com/chroma-core/chroma>

Table 12: Assignments of hyper-parameters for inference.

Parameter	Value
model	gpt-4o-2024-05-13
temperature	0.7
top_p	0.95
n	1
stop	[“\nObservation”, “\nThought”]
max_tokens	512

6.4 Experimental Results on Spider 2.0-lite

Table 13: Statistics of Spider 2.0-lite (SQLite and Snowflake)

Subtask	#Item	#DB	Tb/DB	Col/Tb	FK/DB
SQLite	135	30	13.8	7.0	5.9
Snowflake	198	51	136.6	21.8	0.0

Table 14: Results of Spider 2.0-lite (SQLite and Snowflake)

Method	SQLite		Snowflake	
	EX	#Token	EX	#Token
DIN-SQL	1.50	8,334.5	1.08	59,382.8
DAIL-SQL (0-shot)	4.55	2,744.4	3.45	26,052.2
Ours (2-shot)	12.69	7,424.6	3.89	17,531.8

Spider 2.0 [14] is a sophisticated text-to-SQL framework designed to handle complex queries across various database systems, encompassing diverse SQL dialects and operations. It includes three benchmarks: Spider 2.0, Spider 2.0-snow, and Spider 2.0-lite, where Spider 2.0-lite involves three database systems—SQLite, Snowflake, and BigQuery. Since these benchmarks focus on various dialects across different databases, which is not the focus of this paper, and considering the cost associated with using Google BigQuery database, we only report the experimental results on the SQLite subset and Snowflake databases subset in Spider 2.0-lite benchmark.

Statistical details in Table 13 show major differences between SQLite and Snowflake settings. Snowflake has about 10 times more tables per database than SQLite (136.6 vs 13.8) and approximately 30 times more columns per database.

Experimental results, summarized in Table 14, demonstrate that our method significantly outperforms the baseline. On SQLite, our method achieves an 8.1-point improvement over DAIL-SQL. While DAIL-SQL uses the entire DB schema as prompt, Spider 2.0’s lack of column descriptions and relatively small schema size (average 96 columns per DB) reduces DAIL-SQL’s burden. Our method consumes more tokens due to multi-turn interactions and the inclusion of 2-shot exemplars.

As Spider 2.0 incorporates complex elements such as Common Table Expressions, grouped aggregations, and SQL dialect variations, the primary performance bottleneck lies in the reasoning capabilities of LLMs rather than challenges associated with wide tables.

When presented with a question, you need to use specific tools to interact with a locally deployed SQLite database and craft an SQL query to retrieve the answer. Below are descriptions of the tools and some essential usage patterns:

1. **SearchValue**(query: Union[str, List[str]], table: Union[str, List[str]] = [], column: Union[str, List[str]] = [])

Description: This tool searches for a value in the database based on the 'query' parameter. It utilizes Elasticsearch to find the most relevant cell value, and returns the actual value, the corresponding column name, and the table name. You can also specify the 'table' and 'column' parameters to search within a specific table or column. If these parameters are not specified, the tool will search across all tables and columns.

Note: This tool does not index the id column because the id column is not a text field.

2. **SearchColumn**(query: Union[str, List[str]])

Description: This tool searches for columns in the database based on the 'query' parameter, which indicates the expected column semantics. It returns the column names, types, corresponding tables, descriptions, and value descriptions.

Example: If you want to identify the column for "population in the year 2020," you can use: SearchColumn("population in year 2020"). This will return: {'name': 'population_2020', 'format': 'INTEGER', 'table': 'zip_data', 'column_description': 'the population of the residential area in 2020', 'statistics': 'distinct count: 15389'}.

Note: The 'statistics' field varies depending on the format. If it starts with 'text field', several examples of values will be listed, and you should use the 'SearchValue' tool to find the specific value you need. If it is an enumerable type, it will count the number of occurrences. If it is a numeric type, the maximum and minimum values will be provided.

3. **FindShortestPath**(start: Union[str, List[str]], end: Union[str, List[str]])

Description: This function identifies the shortest path between two sets of columns in a database schema, treated as an undirected linked graph. In this graph, nodes represent columns, edges represent relationships through column names (linking columns to tables) and foreign key constraints (linking tables). The parameters 'start' and 'end' accept either a single column or a list of columns specified in the '{table}. {column}' format. The function returns the shortest path between each possible pair of start and end nodes, which can be utilized to construct SQL queries.

Example: If the query requires the city name and the population of a city for a specific year, you might call 'FindShortestPath(start=[t1.city", "t2.population"], end="t3.year")'. This would return paths such as '[(t1.city', 't3.year', 't1.city <-> ... t3.year'), ('t2.population', 't3.year', 't2.population <-> ... t3.year')]'.

4. **ExecuteSQL**(sql)

Description: Executes an SQL query. This tool enables the execution of any SQL query within the database.

General Process:

- If a question includes a specific string value or entity name, utilize the SearchValue tool to identify the corresponding cell value, column name, and table name.
- Use SearchColumn to locate the column that is semantically related to the query.
- Determine the shortest path between the selected columns and the condition columns using FindShortestPath
- Execute the SQL query using ExecuteSQL. The result from the final call to ExecuteSQL will be taken as the answer, so it is critical to ensure that the SQL query is accurately formulated and complete.

Crucial Patterns:

- You need to use your intelligence and reasoning to determine how columns in the database reflect the semantics of the question, then use SearchColumn() to locate them. For instance, "the city Bishopville, SC" may be represented by two columns: T.city = 'Bishopville' AND T.state = 'SC'.
- Be aware of dirty data in the database. Use 'IS NOT NULL' to filter out missing values.
- MUST construct SQL queries strictly according to the observed values. For example, 'Akiak' includes a trailing space.
- DO NOT select irrelevant columns. (e.g. for the question "who has the max value", you should NOT return the max value itself, but if the question is "what is the max value, list the people", you should return the max value and the corresponding people.)
- Use numbers from 0 to 100 to represent percentages. For example, use *100 in SQL if necessary.
- DO NOT concatenate column in select statement. For example, 'SELECT c1 || '' || c2' is not allowed. You should SELECT c1, c2 instead.

Remember:

- DO NOT fabricate column names. Column names must be based on actual observation.
- Follow the **exact format** provided in the exemplars: specify an action after 'Action:', only one at a time, and the action MUST be one of [SearchColumn, SearchValue, FindShortestPath, ExecuteSQL, Done].
- STOP after the action to conclude this round of interaction. For example: Action: SearchColumn(...)
- Please keep interactions concise and accurate, avoiding unnecessary details.

Figure 3: Prompt text.

6.5 Experimental Results with Open-source LLMs

Table 15: Results on Spider-Dev and its variants with Open-source LLMs

Method	Spider-Dev		Spider-DK	
	EM	EX	EM	EX
GPT-4o	52.0	84.8	41.5	75.5
Meta-Llama-3.1-8B-Instruct	21.4	40.2	21.9	46.4
Minstral-8B-Instruct-2410	28.1	51.4	28.6	50.5
Spider-Syn		Spider-Realistic		
GPT-4o	44.9	78.7	55.7	81.3
Meta-Llama-3.1-8B-Instruct	11.5	25.4	32.1	54.3
Minstral-8B-Instruct-2410	27.1	49.9	34.4	55.9

Table 16: Results on BIRD-Dev and BIRD Mini-Dev (SQLite) with Open-source LLMs

Method	BIRD-Dev		Mini-Dev	
	EX	EX	EX	Soft F1
GPT-4o	54.56	46.60	50.75	
Meta-Llama-3.1-8B-Instruct	30.96	20.40	25.94	
Minstral-8B-Instruct-2410	28.49	20.80	23.48	

We conducted experiments using two open-source LLMs, Meta-Llama-3.1-8B-Instruct and Minstral-8B-Instruct-2410, with results shown in Tables 15 and 16. The results indicate that there remains a considerable performance gap between open-source LLMs and closed-source LLMs.

7 GenAI Usage Disclosure

This paper used the web-based version of ChatGPT (accessed via OpenAI's website) solely for grammar corrections and text polishing of the manuscript. No other generative AI tools were used in the research, data collection, analysis, or writing of this paper.

References

- [1] Xinyun Chen, Maxwell Lin, Nathanael Schärlí, and Denny Zhou. 2024. Teaching Large Language Models to Self-Debug. In *The Twelfth International Conference on Learning Representations*. <https://openreview.net/forum?id=KuPixlqPiq>
- [2] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-Grounded Pretraining for Text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 1337–1350. doi:10.18653/v1/2021.naacl-main.105
- [3] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A Survey on In-context Learning. arXiv:2301.00234 [cs.CL] <https://arxiv.org/abs/2301.00234>
- [4] Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021. 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)*. Association for Computational Linguistics, Online, 2505–2515. doi:10.18653/v1/2021.acl-long.195
- [5] Yujian Gan, Xinyun Chen, and Matthew Purver. 2021. Exploring Underexplored Limitations of Cross-Domain Text-to-SQL Generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 8926–8931. doi:10.18653/v1/2021.emnlp-main.702
- [6] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. Proc. VLDB Endow. 17, 5 (may 2024), 1132–1145. doi:10.14778/3641204.3641221
- [7] Yu Gu, Yiheng Shu, Hao Yu, Xiao Liu, Yuxiao Dong, Jie Tang, Jayanth Srinivas, Hugo Latapie, and Yu Su. 2024. Middleware for LLMs: Tools Are Instrumental for Language Agents in Complex Environments. arXiv:2402.14672 [cs.CL] <https://arxiv.org/abs/2402.14672>
- [8] Chunxi Guo, Zhiliang Tian, Jintao Tang, Shasha Li, Zhihua Wen, Kaixuan Wang, and Ting Wang. 2023. Retrieval-Augmented GPT-3.5-Based Text-to-SQL Framework with Sample-Aware Prompting and Dynamic Revision Chain. In *Neural Information Processing: 30th International Conference, ICONIP 2023, Changsha, China, November 2023–23, 2023, Proceedings, Part VI* (Changsha, China). Springer-Verlag, Berlin, Heidelberg, 341–356. doi:10.1007/978-99-8076-5_25
- [9] Chunxi Guo, Zhiliang Tian, Jintao Tang, Pancheng Wang, Zhihua Wen, Kang Yang, and Ting Wang. 2023. Prompting GPT-3.5 for Text-to-SQL with Desemanticization and Skeleton Retrieval. In *PRICAI 2023: Trends in Artificial Intelligence: 20th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2023, Jakarta, Indonesia, November 15–19, 2023, Proceedings, Part II* (Jakarta, Indonesia). Springer-Verlag, Berlin, Heidelberg, 262–274. doi:10.1007/978-99-7022-3_23
- [10] Aric Hagberg, Pieter J. Swart, and Daniel A. Schult. 2008. Exploring network structure, dynamics, and function using NetworkX. Los Alamos National Laboratory (LANL), Los Alamos, NM (United States). <https://www.osti.gov/biblio/960616>
- [11] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. arXiv:2406.08426 [cs.CL] <https://arxiv.org/abs/2406.08426>
- [12] Jinbiao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Singapore, 9237–9251. doi:10.18653/v1/2023.emnlp-main.574
- [13] Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. MCS-SQL: Leveraging Multiple Prompts and Multiple-Choice Selection For Text-to-SQL Generation. arXiv:2405.07467 [cs.CL] <https://arxiv.org/abs/2405.07467>
- [14] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2024. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. arXiv:2411.07763 [cs.CL] <https://arxiv.org/abs/2411.07763>
- [15] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? arXiv:2406.01265
- [16] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. 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/EEAII'23)*. AAAI Press, Article 1466, 9 pages. doi:10.1609/aaai.v37i11.26535
- [17] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodesS: Towards Building Open-source Language Models for Text-to-SQL. Proc. ACM Manag. Data 2, 3, Article 127 (may 2024), 28 pages. doi:10.1145/3654930
- [18] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhu Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2024. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) (NIPS '23). Curran Associates Inc., Red Hook, NY, USA, Article 1835, 28 pages.
- [19] Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. 2024. PET-SQL: A Prompt-Enhanced Two-Round Refinement of Text-to-SQL with Cross-consistency. arXiv:2403.09732 [cs.CL] <https://arxiv.org/abs/2403.09732>
- [20] Linyong Nan, Yilun Zhao, Weijun Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023. Enhancing Text-to-SQL Capabilities of Large Language Models: A Study on Prompt Design Strategies. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, Singapore, 14935–14956. doi:10.18653/v1/2023.findings-emnlp.996
- [21] Ansong Ni, Sriniv Iyer, Dragomir Radev, Ves Stoyanov, Wen-tau Yih, Sida I. Wang, and Xi Victoria Lin. 2023. LEVER: learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) (ICML '23). JMLR.org, Article 1086, 23 pages.
- [22] OpenAI. 2023. GPT-4 Technical Report. ArXiv abs/2303.08774 (2023). <https://arxiv.org/abs/2303.08774>
- [23] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, Vol. 35. Curran Associates, Inc., 27730–27744. https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf
- [24] Mohammadmreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Thirty-seventh Conference on Neural Information Processing Systems*. <https://openreview.net/forum?id=p53QDxSlc5>
- [25] Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. arXiv:2405.15307 [cs.CL] <https://arxiv.org/abs/2405.15307>
- [26] Tonghui Ren, Yunkai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X. Sean Wang. 2024. PURPLE: Making a Large Language Model a Better SQL Writer. arXiv:2403.20014 [cs.DB] <https://arxiv.org/abs/2403.20014>
- [27] Zhili Shen, Pavlos Vougiouklis, Chenxin Diao, Kaustubh Vyas, Yuanyi Ji, and Jeff Z. Pan. 2024. Improving Retrieval-augmented Text-to-SQL with AST-based Ranking and Schema Pruning. arXiv:2407.03227 [cs.CL] <https://arxiv.org/abs/2407.03227>
- [28] Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. 2022. Natural Language to Code Translation with Execution. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 3533–3546. doi:10.18653/v1/2022.emnlp-main.231
- [29] Luohu Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. 2024. Keep the Cost Down: A Review on Methods to Optimize LLM's KV-Cache Consumption. arXiv:2407.18003 [cs.CL] <https://arxiv.org/abs/2407.18003>
- [30] Ruoxi Sun, Sercan Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and Tomas Pfister. 2024. SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL (extended). arXiv:2306.00739 [cs.CL] <https://arxiv.org/abs/2306.00739>
- [31] Zhao Tan, Xiping Liu, Qing Shu, Xi Li, Changxuan Wan, Dexi Liu, Qizhi Wan, and Guoqiong Liao. 2024. Enhancing Text-to-SQL Capabilities of Large Language Models through Tailored Promptings. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. ELRA and ICCL, Torino, Italia, 6091–6109. <https://aclanthology.org/2024.lrec-main.539>

- [32] Dingzirui Wang, Longxu Dou, Xuanliang Zhang, Qingfu Zhu, and Wanxiang Che. 2024. Improving Demonstration Diversity by Human-Free Fusing for Text-to-SQL. arXiv:2402.10663 [cs.CL] <https://arxiv.org/abs/2402.10663>
- [33] Niklas Wretblad, Fredrik Riseby, Rahul Biswas, Amin Ahmadi, and Oskar Holmström. 2024. Understanding the Effects of Noise in Text-to-SQL: An Examination of the BIRD-Bench Benchmark. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 356–369. doi:10.18653/v1/2024.acl-short.34
- [34] Guanming Xiong, Junwei Bao, and Wen Zhao. 2024. Interactive-KBQA: Multi-Turn Interactions for Knowledge Base Question Answering with Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 10561–10582. <https://aclanthology.org/2024.acl-long.569>
- [35] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=WE_vluYUL-X
- [36] 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*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun’ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3911–3921. doi:10.18653/v1/D18-1425
- [37] Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. ACT-SQL: In-Context Learning for Text-to-SQL with Automatically-Generated Chain-of-Thought. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, Singapore, 3501–3532. doi:10.18653/v1/2023.findings-emnlp.227
- [38] Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic Evaluation for Text-to-SQL with Distilled Test Suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 396–411. doi:10.18653/v1/2020.emnlp-main.29