FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis

Chao Zhang
Zhejiang University
Hangzhou, China
zjuzhangchao@zju.edu.cn

Yuren Mao
Zhejiang University
Hangzhou, China
yuren.mao@zju.edu.cn

Yijiang Fan
Zhejiang University
Hangzhou, China
yijiangfan@zju.edu.cn

Yu Mi
Zhejiang University
Hangzhou, China
miyu@zju.edu.cn

Yunjun Gao
Zhejiang University
Hangzhou, China
gaoyj@zju.edu.cn

Lu Chen
Zhejiang University
Hangzhou, China
luchen@zju.edu.cn

Dongfang Lou
Hundsun Technologies INC.
Hangzhou, China
loudongfang2022@gmail.com

Jinshu Lin
Hundsun Technologies INC.
Hangzhou, China
linjs13607@hundsun.com

ABSTRACT

Text-to-SQL, which provides zero-code interface for operating relational databases, has gained much attention in financial analysis; because financial professionals may not be well-skilled in SQL programming. However, until now, there is no practical Text-to-SQL benchmark dataset for financial analysis, and existing Text-to-SQL methods have not considered the unique characteristics of databases in financial applications, such as commonly existing wide tables. To address these issues, we collect a practical Text-to-SQL benchmark dataset and propose a model-agnostic Large Language Model (LLMs)-based Text-to-SQL framework for financial analysis. The benchmark dataset, BULL, is collected from the practical financial analysis business of Hundsun Technologies Inc., including databases for fund, stock, and macro economy. Besides, the proposed LLMs-based Text-to-SQL framework, FinSQL, provides a systematic treatment for financial Text-to-SQL from the perspectives of prompt construction, parameter-efficient fine-tuning and output calibration. Experiments on BULL demonstrate that FinSQL achieves state-of-the-art performance at low cost, and it brings up to 36.64% improvement in few-shot cross-database scenarios.

CCS CONCEPTS

• **Information systems** → **Structured Query Language**.

KEYWORDS

Text-to-SQL, Large Language Models (LLMs), Financial Analysis

ACM Reference Format:

Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. 2024. FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis. In *Companion of the 2024 International Conference on Management of Data (SIGMOD-Companion '24), June 9–15, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3626246.3653375

SIGMOD-Companion '24, June 9–15, 2024, Santiago, AA, Chile

ACM ISBN 979-8-4007-0422-2/24/06.
https://doi.org/10.1145/3626246.3653375

1 INTRODUCTION

Text-to-SQL aims to transform natural language questions into executable SQL queries, which enables low-code operations for relational databases. It can facilitate the data access procedure for non-professional database users who are not familiar with SQL and has gained much attention in various areas, especially in financial analysis. While financial professionals (e.g., investment advisors) need to query relevant databases frequently, they are usually not well-skilled in SQL programming. Therefore, Text-to-SQL is significantly important for financial analysis and has gained much attention.

However, there is no Text-to-SQL benchmark dataset for financial analysis, and existing Text-to-SQL methods have not considered the unique characteristics of databases used in financial analysis. To address these issues, we construct a practical Text-to-SQL dataset for financial analysis based on the intelligent investment assistant product of Hundsun Technologies Inc., which facilitates more than 50 financial institutions (including Alipay, China Merchants Bank, and so on) and serves millions of personal users. This dataset, dubbed BULL, contains three databases corresponding to fund, stock, and macro economy respectively. Besides, in this dataset, there are 4,966 natural language question-SQL pairs annotated by financial professionals, data scientists, and software engineers from Hundsun Technologies Inc. Furthermore, BULL has both English and Chinese versions.

Compared with the widely used Text-to-SQL benchmark datasets (e.g., Spider[39] and BIRD [21]), BULL has much more tables for each database and much more columns for each table, illustrating as in Table 1. Furthermore, table and column names in BULL are often expressed with abbreviations or vague representations. These characters require financial Text-to-SQL models to support large input context length and have strong context understanding ability. Fortunately, Large Language Models (LLMs)-based Text-to-SQL can satisfy these requirements, and several LLMs-based Text-to-SQL methods have been proposed recently. However, existing state-of-the-art LLMs-based Text-to-SQL methods typically depend on OpenAI's APIs, such as GPT-3.5-turbo or GPT-4, which are expensive and have risks of information leakage. Therefore, these methods cannot be used in the financial applications where the information privacy is critically important.

To avoid information leakage, a feasible way is to adopt open-source LLMs (e.g., LLaMA[32] and Baichuan [37]) and train them in

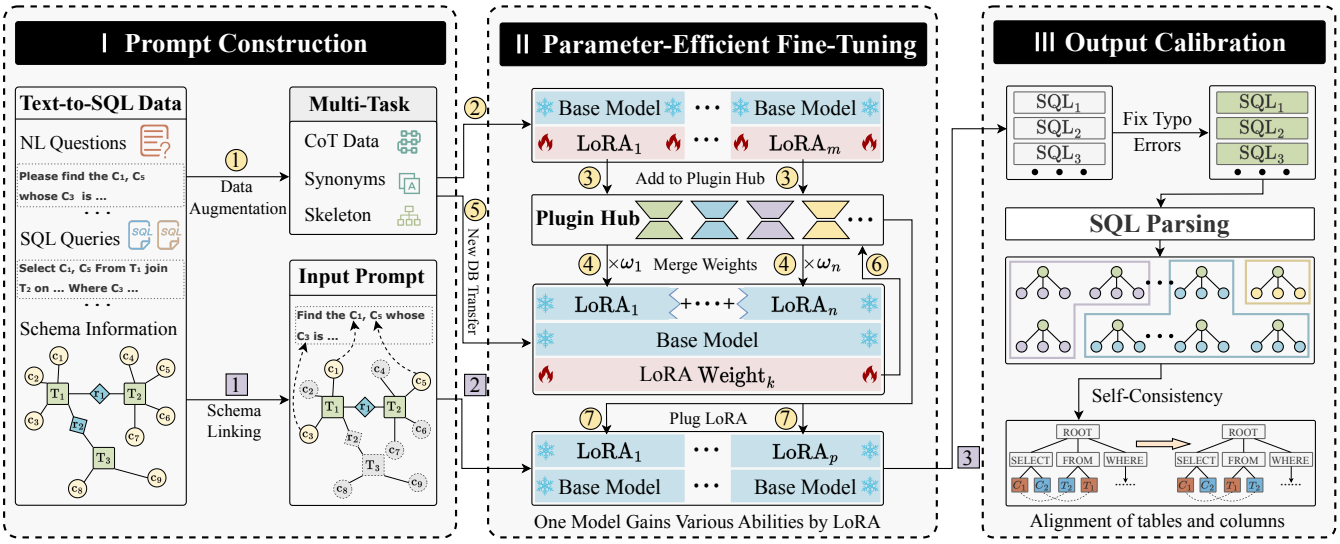

Figure 1: The overview of FinSQL framework. In the training stage, the training data is first augmented with a hybrid data augmentation method. Then, the augmented data is used to train LoRA plugins which are used to handle various Text-to-SQL tasks. The LoRA plugins are managed by a LoRA plugin hub. In the inference stage, schema linking is firstly conducted to obtain a concise prompt, and then the prompt inputs into a LLM model which consists of a base model and a LoRA module constructed with merged LoRA plugins. Finally, the output of the LLM model is calibrated to ensure the correctness of the model output. The ice and fire in the picture mean freezing and updating model weights respectively. The number in and indicate the process step of training and inference respectively.

private domains. However, it faces three challenges: (1) Schema linking dilemma and data scarcity. It is difficult to establish connections between question and schema items for financial databases usually have a large number of columns and tables. Furthermore, due to the labeling cost, the number and diversity of labeled question-SQL pairs are limited. These issues obstruct the construction of concise and diversified prompts, which hinders the model's performance; (2) Resource-consuming fine-tuning and cross-database generalization difficulties. Fine-tuning LLMs on downstream tasks demands several days of computation across multiple GPUs. The substantial cost associated with model updates and iterations poses a considerable challenge. Additionally, transferring the model to a new database incurs significant costs, which impedes the cross-database transfer. (3) Inconsistent output. Due to the inherent randomness and the decoder strategy of sampling, LLMs often generate inconsistent outputs, leading to syntactically incorrect and invalid SQL queries.

To tackle these challenges, this paper proposes a model-agnostic LLMs-based Finacial Text-to-SQL model training and inference framework, dubbed FinSQL [1]. It can be used to develop Text-to-SQL models based on any open-source LLMs. Figure 1 demonstrates the overall overview of FinSQL, which consists of three key components: prompt construction, parameter-efficient fine-tuning, and output calibration, corresponding to the above challenges correspondingly. Specifically, prompt construction consists of a parallel schema linking method and a hybrid data augmentation method, which helps to construct more concise and diverse prompts and enhances the model's performance from the input side.

[1]The dataset and code are available: https://bull-text-to-sql-benchmark.github.io.

Table 1: Differences Between Datasets.

Dataset	Sample Num	Table/DB	Column/DB
WikiSQL [43]	80654	1	6.3
Spider [39]	10181	5.1	27.1
BIRD [21]	12751	7.3	54.2
BULL	**4966**	**26**	**390**

The parameter-efficient fine-tuning component adopts Low-Rank Adaptation (LoRA) to fine-tune a very small percentage of parameters (<1%) to obtain weight plugins for different business scenarios and manages these plugins through a plugin hub. Based on this plugin hub, the database-specific Text-to-SQL models can achieve efficiently few-shot cross-database transfer. In output calibration, SQL post-processing is performed to enhance the correctness of the generated SQL. These three components contribute to FinSQL's superior performance.

Our contributions can be summarized as follows:

- We propose BULL, a practical benchmark dataset for financial Text-to-SQL.
- We propose FinSQL, a model-agnostic LLMs-based Text-to-SQL framework for financial analysis.
- Extensive experimental results on BULL demonstrate that FinSQL is model-agnostic and able to achieve the state-of-the-art performance; furthermore, FinSQL can bring up to 36.64% performance improvement in scenarios requiring few-shot cross-database model transfer.

2 RELATED WORKS

In this section, we introduce various Text-to-SQL datasets. We explore the distinctive features of various datasets and highlight the existing gap between these datasets and real-world financial scenarios. Additionally, we present an overview of recent developments in Text-to-SQL models.

2.1 Text-to-SQL Datasets

High-quality datasets play an important role in the development and evaluation of Text-to-SQL systems. Early Text-to-SQL datasets such as GeoQuery [40] and Scholar [14] provided valuable insights, but they were limited in terms of the number of queries, focused on single-database scenarios, and featured relatively simple SQL queries. WikiSQL [43] and Spider [39], on the other hand, offer over 10,000 SQL queries, enabling cross-database migration scenarios and introducing complex multi-table queries.

Despite these improvements achieved, there is still a considerable gap between these datasets and real-world scenarios. Databases used in real-world applications are often larger and may provide external knowledge to help users understand schema information. KaggleDBQA [16] created 272 SQL queries across eight databases and introduced external knowledge to explain the meaning of columns. BIRD [21] offers a greater number of SQL query samples and larger databases and incorporates external knowledge between questions and database contents, making Text-to-SQL datasets more suitable to real-world situations.

However, these datasets still do not fully satisfy the demands of the industry, which are often more complex and challenging. As shown in Table 1, in real industrial scenarios, the number of tables and columns in databases far exceeds existing open-source datasets. This poses challenges for existing Text-to-SQL methods in terms of incorporating schema information, making it difficult for them to work effectively in this scenario.

2.2 Text-to-SQL Models

Text-to-SQL models have gained significant advancements in recent years [15, 26]. Early rule-based approaches rely on handcrafted templates to generate SQL, showing some effectiveness but being heavily dependent on manually defined rules. They have limited applicability to other scenarios, lack scalability, and struggle with generalization.

To overcome these limitations, researchers develop Text-to-SQL methods based on the Seq2Seq architecture. IRNet [10] uses an encoder to represent questions and schema, while RAT-SQL [33], LGESQL [1], and S^2SQL [13] employ graph neural networks to capture alignment relationships between questions and schema, and then use a decoder to generate SQL queries.

Subsequently, with the development of large language models, pre-trained models like T5, mT5, LLaMA, and others demonstrated enhanced language understanding and generalization capabilities. Methods based on fine-tuning these models achieved better results in Text-to-SQL tasks. Graphix [20] enables T5 with multi-hop reasoning ability. Picard [30] uses a constrained decoder to enhance the quality of generated SQL queries. RESDSQL [19] uses a two-stage approach to first retrieve schema elements relevant to the question and then generate the corresponding SQL query, which is now the SOTA fine-tuning-based method in Spider leaderboard.

More recently, large language models (LLMs) have exhibited remarkable capabilities and left a deep impression. With the In-context learning technique, these models can understand human instructions and execute tasks without requiring retraining. Currently, leading approaches on the Spider leaderboard are based on GPT models using In-context Learning, such as C3, DIN-SQL, and DAIL-SQL. However, these methods involve invoking the OpenAI API, which can be costly and pose privacy risks. Furthermore, instruction templates need redesigning for different scenarios.

2.3 Chain of Thought

Chain of Thought (CoT) [34] is a method that enables LLMs to think like humans to solve complex reasoning tasks. In the CoT setting, LLMs first output the reasoning progress and then provide the final answer, which demonstrates substantial improvements in handling reasoning tasks. In the challenging arithmetic reasoning benchmark GSM8K, the CoT technique has led to a substantial improvement in Google's large language model, Palm-540B [2]. The accuracy on this benchmark increases significantly, rising from 17.9% to 56.5% [34]. Additionally, the CoT technique instructs large language models (LLMs) to first generate the reasoning process and then produce the final result. This process enhances the interpretability of the model's output.

2.4 Parameter-Efficient Fine-Tuning

Parameter-Efficient Fine-Tuning (PEFT) methods (e.g., Adapter [11], Prompt Tuning [18], Prefix-Tuning [22], LoRA [12]), which only tunes a small fraction of parameters of pre-trained LLMs and can achieve comparable performance with full-parameter fine-tuning, enables an LLM to adapt to downstream tasks with few computational resources. They only need to fine-tune a small fraction of the model weights (<1%) to adapt it to downstream tasks. This process can be completed in a short time on a single GPU.

3 PRELIMINARIES

In this section, we formalize the problem of Text-to-SQL for finance and introduce the related techniques employed in our proposed method, including data augment based on LLMs, schema linking, large language models, low-rank adaption, and self-consistency.

3.1 Problem Formulation

This paper focuses on LLM-based Text-to-SQL systems. In this section, we first give the definition of Text-to-SQL and then introduce the LLM-based Text-to-SQL systems.

Text-to-SQL. Given a natural language query Q and its corresponding database schema $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{R})$ where $\mathcal{T} = (t_1, t_2, ...)$, $\mathcal{C} = (c_1, c_2, ...)$, $\mathcal{R} = (r_1, r_2, ...)$, which present multiple tables, columns and foreign keys relations respectively, a Text-to-SQL system aims to generate a executable SQL query $\mathcal{Y}$ corresponding to Q. Traditional Text-to-SQL methods typically adopt the encoder-only or encoder-decoder models (e.g. SyntaxSQLNet [38], LGESQL [1]). Recently, benefiting from the emergent abilities of the decoder-only Large Language Models (LLMs), LLM-based Text-to-SQL methods

achieve the state-of-art performance. Next, we introduce the working mechanism of the LLM-based Text-to-SQL system.

LLM-based Text-to-SQL System. LLM-based Text-to-SQL utilizes LLMs $\mathcal{M}$ to generate SQL queries. It begins by receiving an instruction prompt, denoted as $\mathcal{P}(Q, \mathcal{S})$, which is generated by combining a question Q and a database schema $\mathcal{S}$ into a prompt template. Subsequently, the model $\mathcal{M}$ estimates the probability distribution over SQL query $\mathcal{Y}$ and generates it token by token. The generation process for the SQL query $\mathcal{Y}$ can be formulated as follows:

$$P_{\mathcal{M}}(\mathcal{Y}|\mathcal{P}(Q,\mathcal{S})) = \prod_{i=1}^{|\mathcal{Y}|} P_{\mathcal{M}}(\mathcal{Y}_i|\mathcal{P}(Q,\mathcal{S}), \mathcal{Y} < i) \quad (1)$$

3.2 Schema Linking

Schema linking, which aims to link the meta data of a database schema to a natural language query, is a fundamental component in the Text-to-SQL system[17]. Specifically, given a natural language query Q and its corresponding database schema $\mathcal{S} = (\mathcal{T}, C, \mathcal{R})$, schema linking extracts a subset of meta data that is relevant Q. The results of schema linking is denoted as $\mathcal{S}' = (\mathcal{T}', C', \mathcal{R}')$, where $\mathcal{T}' \in \mathcal{T}, C' \in C'$ and $\mathcal{R}' \in \mathcal{R}$. Schema linking can help to reduce LLMs' misunderstanding about the natural language query by reducing the noise caused by redundant tables and columns. Consequently, it can effectively enhance the quality of generated SQL queries.

Early Text-to-SQL methods, exemplified by IRNet and IESQL, rely on rule-based or character-matching techniques to perform exact or partial matches between the tokens in the question and the elements in the database schema. Subsequently, they identified the relationships between the query and the table and column names, which are then leveraged to generate the SQL query corresponding to the question. However, the generalization ability of these approaches is limited. When encountering words with similar meanings, they often fail to make accurate matches. Furthermore, these methods cannot capture the comprehensive semantic relationships between natural language and table structures, as well as the various relationships among schema elements. To express this structural relationship, some researchers employ graph neural networks (GNN) for representation, such as LGESQL, RATSQL, and others, which have yielded promising results. However, These approaches necessitate the use of specific GNN modules as part of the overall model and are difficult to adapt to prevalent open-source large models such as LLaMA, Baichuan, ChatGLM, etc.

To enhance compatibility with LLMs, some methods isolate schema linking as a separate module. RESDSQL utilizes the Roberta model as a base model to build a Cross-Encoder model to recall and rank schema elements. C3 and DIN-SQL provide well-designed in-context learning instructions to the GPT models for retrieving relevant schema items. These methods achieve outstanding performance on popular benchmarks, such as spider leaderboard.

3.3 Cost of LLMs

Different large models have varying context lengths, typically measured in terms of the number of tokens. In OpenAI, GPT-4 offers context lengths of 8k and 32k tokens. The efficiency of the tokenizer for GPT-4 and ChatGPT is approximately such that 1000 tokens correspond to around 700 English words. As a result, the amount of information a large model can handle at once is limited.

Table 2: API Price of GPT Models

Model	Input	Output
GPT-4-8k	\$0.03 / 1K tokens	\$0.06 / 1K tokens
GPT-4-32k	\$0.06 / 1K tokens	\$0.12 / 1K tokens
GPT-3.5-turbo-1106	\$0.001 / 1K tokens	\$0.002 / 1K tokens

We can access the ChatGPT and GPT-4 models by utilizing their respective paid API interfaces. The pricing details for these models are provided in Table 2.

3.4 Hallucination of LLMs

The phenomenon of hallucination in LLMs refers to the situation that LLMs, when generating factual content, sometimes produce information that appears to be correct but contradicts actual facts. The essence of this phenomenon lies in the LLMs' inability to maintain precise control over knowledge [41]. Even the most powerful LLMs like GPT-4 may still encounter this issue.

3.5 Uncertainty of LLMs

The output of LLMs exhibits uncertainty and instability due to the inherent randomness within them [23]. LLMs generate multiple results when executed multiple times with the same input prompt. This issue may result in the generation of a correct SQL query one time, but an incorrect one the next time. Therefore, it is of great significance for Text-to-SQL to enhance the consistency of the generated output content from LLMs.

4 OVERVIEW

Large Language Model (LLM)-based Text-to-SQL methods for financial analysis typically encounter three challenges: (1) Schema linking dilemma and data scarcity. Databases for finance applications usually consist of a large number of wide tables that contain lots of columns, which makes it hard to establish connections between question and schema items. Furthermore, the number and diversity of labeled question-SQL pairs are limited. These issues obstruct the construction of concise and diversified prompts, which hinders the model's performance; (2) Resource-consuming fine-tuning and cross-database generalization difficulties. Fine-tuning LLMs on downstream tasks demands several days of computation across multiple GPUs. The substantial cost associated with model updates and iterations poses a considerable challenge. Additionally, transitioning the model to new database scenarios incurs significant costs. (3) Inconsistent output. Due to the inherent randomness and the decoder strategy of sampling, LLMs often generate inconsistent outputs, which leads to syntactically incorrect and invalid SQL queries.

To tackle these challenges, we propose a Text-to-**SQL** framework for **Fin**ancial data, **FinSQL**, which consists of three key components addressing the challenges correspondingly. Figure 1 provides an overview of the proposed FinSQL, which consists of three key

components, **Prompt Construction**, **Parameter-Efficient Fine-Tuning**, and **Output Calibration**. The Prompt Construction module proposes a parallel Cross-Encoder model to retrieve relevant schema elements and a hybrid data augmentation method to enrich the original dataset, helping to construct clear prompts and enrich training data. The Parameter-Efficient Fine-Tuning module utilizes the augmented dataset to fine-tune LLMs with LoRA method, which can significantly reduce the computational cost. This module also provides a LoRA weight merging method for further training, which significantly improves the performance of LLMs when transferring across different databases. The Output Calibration designs an algorithm that performs SQL syntax checking and self-consistency to improve the overall consistency and coherence of the final output SQL queries. The details of these modules are introduced as follows.

Prompt Construction. This module focuses on constructing clear prompts and enriching the training data, which are beneficial to both the training and inference processes. It augments the original data based on LLMs and well-designed rules, resulting in a new dataset that covers three SQL-related instruction prompts: chain-of-thought data, synonymous question data, and skeleton data. The augmented data is later used to fine-tune LLMs, which can significantly improve the model's reasoning and comprehensive ability in Text-to-SQL compared to the original data. Besides, this module trains a Cross-Encoder model for retrieving relevant schema elements corresponding to the question. It can help eliminate irrelevant elements and generate more concise and clear prompts, which leads to significant improvement in model performance.

Parameter-Efficient Fine-tuning. Provided with the augmented data, this module uses the Parameter-Efficient Fine-Tuning (PEFT) methods, such as LoRA, to adapt LLMs to downstream Text-to-SQL task on a single GPU in a short time. These trained LoRA weights are then added to the LoRA plugin Hub. In the case of low-resource database scenarios, the original parameters of the base model are first updated by merging them with previous LoRA weights from Plugin Hub. Then, an additional LoRA module is applied for further fine-tuning on this foundation. This database domain transfer approach demonstrates the ability to achieve strong performance with limited data. A database corresponds to a LoRA module, which stores in the LoRA Plugin Hub. When dealing with different database scenarios, we can effortlessly plug the relevant LoRA module into the base model to generate SQL queries.

Output Calibration. Due to the inherent randomness within LLMs, the SQL queries LLMs generate are often uncertain and may not strictly adhere to the SQL syntax rules, resulting in invalid SQL queries. This module incorporates a post-processing augmentation algorithm specifically tailored for SQL, which leads to more consistent SQL queries. It initially corrects typo errors in SQL and subsequently parses the SQL queries to extract keywords and values, which are then utilized to mine the most consistent SQL. At last, it employs alignment between tables and columns for the SQL queries.

5 BULL: FINANCIAL TEXT-TO-SQL DATASET

BULL consists of three databases corresponding to fund, stock, and macro economy respectively. Besides, there are 4,966 natural language questions-SQL pairs annotated by financial professionals, data scientists, and software engineers from Hundsun Technologies Inc. In this section, we introduce the details about the databases and the process of annotation.

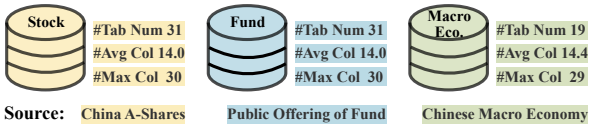

Figure 2: The introduction of BULL databases. The English and Chinese versions of BULL share the same database structure. #Tab Num represents the number of tables in the database. #Avg Col and #Max Col mean the average and maximum number of columns in each table within the database.

Question: Which funds have a fund establishment size exceeding 1 billion? And what is their three-year annualized return rate?

SQL Query:
```
SELECT a.secuabbr, a.annualizedrrinthreeyear
FROM mf_netvalueperformancehis as a JOIN mf_fundarchives as b
ON a.innercode = b.innercode
Where b.foundedsize > 1000000000;
```

Table Name & Column Name	Table and Column Description
mf_netvalueperformancehis	Latest range performance of public fund net value
secuabbr	Fund abbreviation
annualizedrrinthreeyear	Three-year annualized return rate (%)
mf_fundarchives	Public fund overview
foundedsize	Fund establishment size (units)

Figure 3: An example of BULL dataset

5.1 Database Source

BULL consists of three databases that build based on the intelligent investment assistant product of Hundsun Technologies Inc. The three databases are constructed with data related to China A-Shares, China Public Offering of Fund, and Chinese Marco Economy. The three databases contain 31, 28, and 19 tables respectively. The cutoff date of the including data is up to April 2022. Most of the tables have more than ten columns. Figure 2 illustrates the details of this dataset. Furthermore, table and column names in BULL are often expressed using abbreviations or vague representations. One example of the data in BULL is shown in Figure 3.

5.2 Annotation

Generating question-SQL pairs for training Text-to-SQL models requires professional financial knowledge. To construct question-SQL pairs for BULL, several financial professionals, data scientists and software engineers of Hundsun Technologies Inc. work together for more than one week. The financial professionals and data scientists first write 4966 diverse Chinese questions. Then, the software engineers write corresponding SQL queries for the questions. Subsequently, three interns with both knowledge of database and finance validate the correctness of these question-SQL pairs. Furthermore, they extend BULL to an English version. Specifically, they translate the descriptions of tables and columns using professional financial terminology. Then, two interns translate the questions, while the other intern scrutinizes the translated sentences for grammatical

correctness and adherence to customary financial expressions. Finally, they rewrite the SQL queries to suit English databases. This process involves replacing the Chinese values with their English counterparts mentioned in the questions. The above steps result in the creation of the comprehensive BULL dataset.

6 PROMPT CONSTRUCTION

Prompts significantly impact the performance of LLMs-based Text-to-SQL. To construct proper prompts, this paper proposes a hybrid data augmentation strategy and a parallel schema linking method. The hybrid data augmentation strategy jointly leverages LLMs and carefully designed rules to enrich the training data. This process can improve both the quality and quantity of the training data, thereby improving the performance of the models trained in the subsequent steps. Besides, the parallel schema linking method adopts a Cross-Encoder model to retrieve several schema elements relevant to the question in parallel, which can effectively reduce the irrelevant content and noise for the input data to the model. It ultimately assists the model in generating SQL queries with higher quality.

6.1 Hybrid Data Augmentation

In finance applications, data labeling requires professional knowledge and numerous manual efforts, which results in a shortage of professional annotators and constrained labeled data, leading to insufficient data and a lack of data diversity. It is necessary to enhance the diversity of the dataset because a diverse dataset reflects a broader range of real-world scenarios and conditions.

Data augmentation is a natural choice to enrich the dataset. However, existing Text-to-SQL augmentation methods primarily focus on enhancing SQL queries and questions [5], without considering the reasoning process underlying the generation of SQL. With the development of LLMs, more and more data augmentation methods about reasoning ability based on LLMs have emerged[4], which achieves excellent results. In this paper, we propose a hybrid data augmentation method consisting of three data forms, denoted as chain-of-thought, synonymous questions, and skeleton SQL, which correspond to the reasoning ability, meaningful expressions, and SQL structures respectively.

6.1.1 Chain-of-thought Augmentation. The Chain-of-Thought (CoT) prompting technique can effectively improve the reasoning ability of LLMs [34]. Existing methods, such as CoT-KA[35], leverage CoT prompting to instruct GPT-3 in generating CoT reasoning content. Subsequently, it synthesizes new CoT data using this content. However, these methods are unable to apply to Text-to-SQL, because LLMs are likely to generate wrong SQL queries, which can lead to incorrect CoT reason content and is harmful to the model performance. Therefore, it is vital to increase the accuracy of the generated CoT reasoning content.

To address this issue, we design a specialized prompt template that can guide effectively LLMs in generating correct CoT content. Additionally, we use an execution-based self-check module to filter out incorrect content. Figure 4 provides the overview of the CoT generation progress.

For a specific data point extracted from the initial dataset, we first place the SQL query into the corresponding database for execution. If the execution yields empty results, we skip that data. Otherwise,

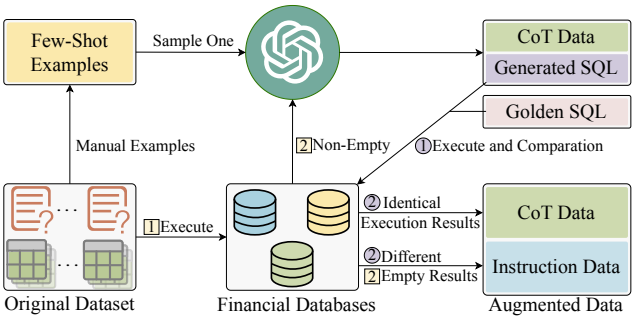

Figure 4: The overview of CoT generation based on self-check

we combine it with a one-shot example to create the prompt. Then the prompt is used to instruct the LLMs to generate intermediate reasoning content, which consists of logical explanations and steps leading to the final output. Finally, we execute both the generated SQL and initial SQL in the database. If the results of these executions match, we deem the generated data as accurate; otherwise, we discard that data.

Here, we first filter out the SQL with empty execution results. This step is essential because when both the golden SQL and generated SQL have empty execution results, it becomes challenging to ensure semantic consistency between these two SQL queries. We also perform a self-check step to exclude the generated SQL queries that have different execution results from the golden SQL query. The success rate comparison of generating CoT with and without self-check can be seen in Table 3.

We design a specific prompt template, where we first give the LLM one example of generating CoT content to specify the desired generation style. Then we provide the golden SQL query and ask the LLM to generate CoT content related to it. Since only the CoT content is required rather than the SQL corresponding to the question, we can get excellent CoT content in this way.

Table 3: Success rate of generating CoT with different methods in Chinese dataset.

Method	Success	Failure	Empty Execution
w self-check	69.14%	18.25%	12.61%
w/o self-check	29.95%	57.44%	12.61%

6.1.2 Synonymous Question Augmentation. A SQL query can correspond to various natural language questions; however, in the labeling process for text-to-SQL, the annotators typically label a SQL query with only one natural language question. Therefore, the training data cannot well support users' distinct linguistic styles, and it is necessary to expand natural language questions to enrich the diversity of question styles.

To enrich the diversity of question styles, we utilize ChatGPT to automatically generate synonymous questions, as shown in Figure 5. We manually write several examples of questions paired with their synonymous sentences as few-shot examples. Subsequently, we

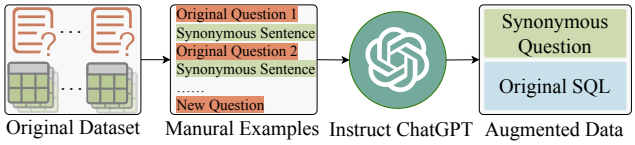

Figure 5: The overview of synonymous question generation

Question	Help me count the number of fund managers with different organizational forms and display them in ascending order.
Golden SQL	select organizationform, count(*) from mf_investadvisoroutline group by organizationform order by count(*) asc;
SQL Skeleton	select _ , count (_) from _ group by _ order by count (_) asc

Figure 6: An example of SQL skeleton

Instruction	Skeleton Instruction + {question} + {Schema}
Skeleton output	**Generated SQL skeleton: ```{skeleton}```** Generated SQL: ```{sql}```

Figure 7: An example of skeleton augmented data

concatenate the provided question to these examples to form a comprehensive prompt. This prompt instructs ChatGPT to generate synonymous questions.

6.1.3 Rule-based Augmentation. In addition to enhancing data quality from the perspective of natural language questions, we can also enhance data from the perspective of the skeleton/structure of SQL queries [9, 19]. The SQL skeleton contains all the SQL keywords with placeholders for missing identifiers, such as table names and column names. One example of the SQL skeleton can be seen in Figure 6.

We design rules for extracting keywords from SQL queries to obtain their corresponding skeletons. Subsequently, we create the skeleton augmented dataset, as shown in Figure 7, which instructs the model to generate SQL skeletons first and then generate the final SQL queries during the training stage.

6.2 Parallel Schema Linking

Schema Linking, which aims to build connections between question and schema items, is important for improving the performance of Text-to-SQL [17]. However, existing state-of-the-art schema linking methods cannot be directly adopted in real-world financial scenarios. Specifically, Graph neural network (GNN)-based approaches [20] are not readily adaptable to widely-used open-source LLMs such as LLaMA, Baichuan, etc; besides, although Cross-Encoder-based methods [19] produce excellent results and can be developed as a separate module independent of LLMs, they are constrained by the context length limitations of BERT-like models and are not well-suited for handling scenarios involving multiple wide tables. Bi-Encoder models can rapidly retrieve information from large volumes of data by constructing embedding indices of

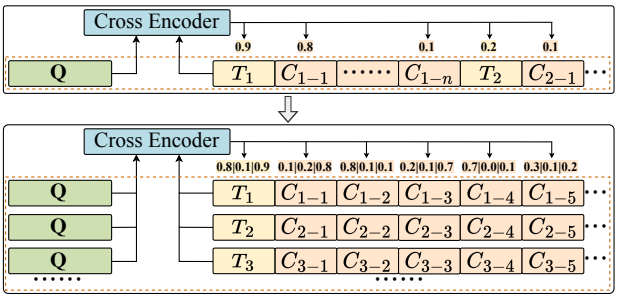

Figure 8: The inference process of Cross-Encoder model

schema items, but their performance typically falls short of that offered by Cross-Encoder models [29].

To perform schema Linking for financial text-to-SQL effectively and accurately, we directly utilize the Cross-Encoder model architecture [19] and improve the model's training and inference procedure to adapt to the financial scenarios, as shown in Figure 8.

The original Cross-Encoder model concatenates all tables and columns and then predicts probability values for each of them in a serial manner. As the number of tables and columns grows, the time required for sequential probability prediction increases linearly, leading to inefficient and slow system performance. Besides, when the number of tables and columns is large, the model cannot load all the schema elements and ultimately fails to complete the text-to-SQL task.

To overcome these issues, we propose a parallel Cross-Encoder model for rapid and accurate retrieval of schema items. Rather than serializing schema items into a single sequence, we organize the tables into a batch, where each element represents a table along with its corresponding column descriptions. The batch, with a size equivalent to the number of tables, is then fed into the modified Cross-Encoder model, enabling it to predict probability values for each table and its associated columns in parallel. This parallel Cross-Encoder model demonstrates the capability to rapidly and accurately retrieve relevant tables and columns from hundreds of schema items.

7 PARAMETER-EFFICIENT FINE-TUNING

Full-parameter fine-tuning, which updates all internal model parameters to handle downstream tasks, faces the following issues: (1) High computational cost. It usually takes extensive training time to fine-tune an LLM across a substantial number of GPUs. (2) High storage cost. After fine-tuning LLMs on downstream tasks, the whole new model parameters are required to be saved, resulting in model weight files of several tens of GBs. (3) Lack of cross-database generalization ability. The full-parameter fine-tuning is typically database-specific, which makes the tuned model difficult to generalize to other databases.

To address these issues, this paper proposes a LoRA-based Parameter Efficient Fine-Tuning framework, which supports low-resource fine-tuning and cross-database generalization. This framework consists of a LoRA-based Parameter-Efficient Fine-Tuning (PEFT) method, a LoRA plugin Hub, and a weights merging method,

which enables low computational cost, low storage cost, and cross-database generalization respectively.

7.1 LoRA-based Multi-Task PEFT

In this section, we propose a multi-task parameter-efficient fine-tuning method based on Low-Rank Adaptation (LoRA) [12]. LoRA only tunes the weights of additional rank decomposition matrices and can achieve comparable performance with full-parameter fine-tuning, which can significantly reduce the computational resources required for fine-tuning large models. Specifically, for a pre-trained weight matrix $W_0 \in R^{d\times k}$, LoRA adds two decomposition matrices on it, denoted as A and B, where $A \in R^{d\times r}, B \in R^{r\times k}$ and $r \ll min(d,k)$. Usually, A is initialized using a Gaussian distribution, while B is initialized with zeros. Then the forward process of the pre-trained weight can be represented like this:

$$h = W_0 x + \Delta W x = W_0 x + BAx \tag{2}$$

During the training process, we freeze the original weights of the base model and solely update the weights of the two low-rank matrices, A and B.

We perform LoRA on three SQL-related instruction tasks, chain-of-thought generation, synonymous question-driven SQL generation, and skeleton-aware generation. Chain-of-thought (CoT) generation task asks the LLMs to output the reasoning process first and then output the final SQL query, which helps to enhance the reasoning ability of LLMs. Synonymous question-driven SQL generation replaces the original question with synonymous questions from ChatGPT to instruct the LLMs to generate SQL queries directly, which enables the LLMs to learn meaningful expressions. Skeleton-aware generation requires the LLMs to first generate the SQL skeleton and then output the SQL query, leading LLMs to a more comprehensive understanding of SQL structures. In these datasets, each table and column concatenates its descriptions, ensuring a comprehensive understanding of the database structure. We combine these datasets by uniformly mixing them together, which are subsequently used to fine-tune the LLMs using the LoRA method.

Figure 9 gives an overview of the proposed LoRA-based multi-task PEFT method. We use datasets for multiple tasks based on data augmentation presented in Section 6.1 and select an open-source LLM as the base model. We freeze the weights of this base model, as indicated by the blue portion in the figure. Subsequently, we plugin a LoRA module, as highlighted in red, and continuously update the parameters of LoRA module using the multi-task dataset. Finally, the trained LoRA module is saved to the Plugin Hub which is introduced in Section 7.2.

7.2 LoRA Plugin Hub

By using the LoRA-based multi-task PEFT method, we can efficiently obtain LoRA modules. Furthermore, the LoRA modules are independent of the base model and have a small size (typically less than 100 MB). Therefore, we can train a series of LoRA modules for different uses. For example, we can train a LoRA module for each database to perform database-specific text-to-SQL. We can also train a LoRA module on all the given databases jointly to obtain cross-database generalization ability. This paper stores these

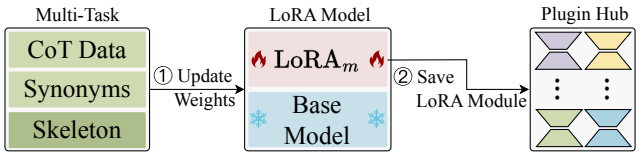

Figure 9: The process of fine-tuning LLMs via LoRA.

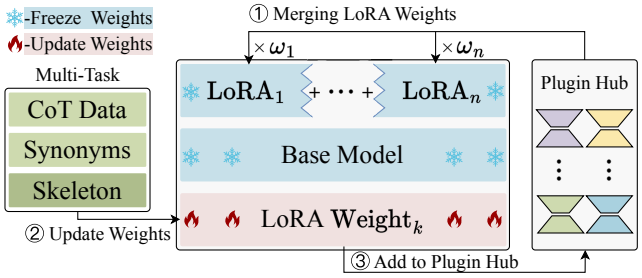

Figure 10: The process of few-shot LoRA-based fine-tuning with weights merging.

LoRA modules together and builds a LoRA module hub, which is dubbed as LoRA plugin hub for the LoRA modules are independent of the base model and can be regarded as plugins for the base model. Besides, the LoRA plugin hub can include LoRA plugins trained on different base models, such as LLaMA2 [32], Baichuan2 [37]. The LoRA plugin hub is able to support few-shot fine-tuning for low-resource scenarios, which is introduced in Section 7.3.

7.3 Weights Merging-based Few-shot LoRA

In practice, it is common that new databases are rapidly constructed with business growth. For a new database, the number of training data is limited; thus, it is necessary to perform few-shot fine-tuning to overcome such a low-resource scenario. This paper proposes a weights merging-based few-shot LoRA-based fine-tuning method to achieve efficient cross-database generalization and handle the text-to-SQL tasks for new databases.

In this paper, the weights merging is conducted over the LoRA plugins stored in the LoRA plugin hub. Specifically, we take several related LoRA plugins from the LoRA Plugin Hub and merge them into a single LoRA plugin by means of weighted summation. The merged LoRA plugin can be represented in the following formula:

$$\hat{A} = \omega_1 A_1 + \omega_2 A_2 + \cdots + \omega_n A_n \tag{3}$$

$$\hat{B} = \omega_1 B_1 + \omega_2 B_2 + \cdots + \omega_n B_n \tag{4}$$

where A_i, B_i $(i \in [1,n])$ are the low-rank matrices of the i-th LoRA plugin, $\hat{A}, \hat{B}$ are the low-rank matrices of the new LoRA plugin, ω_i is the weight coefficient of each LoRA plugin and n is the number of LoRA plugins to be merged.

After that, we initialize the base model with the merged LoRA plugin. Subsequently, we plug an additional LoRA plugin, denoted as A_k, B_k into the base model and further fine-tune it on this foundation. Then the forward process of a pre-trained weight of the

Typo error	select b.secuabbr from mf_managerexperience as a join mf_personalinfo as b on a.personalcode = a.personalcode where b.education == 'bachelor'
Invalid column	select aquirername from lc_legaldistribution where secucode = '603059' and strftime('%Y', infopubldate) = strftime('%Y', DATE('now', '-1 year')) and strftime('%m', infopubldate)='4' order by **aquirementrium** limit 10
Mismatch between table and column	select **b.chinameabbr**, **a.firstindustryname** from **lc_sharestru** as a join **lc_exgindustry** as b on a.companycode = b.companycode where a.totalshares > 10000000000

Figure 11: The examples of invalid SQL queries generated by LLMs. The first SQL makes syntactical mistakes in join and where conditions. The second one uses an invalid column, as column *aquirementrium* is nonexistent in the database. The true column in the golden SQL is *aquireramount*. The third SQL builds the wrong connections between tables and columns. The column *chinameabbr* and *firstindustryname* belong to table *lc_sharestru* and *lc_exgindustry* respectively.

base model can be represented as:

$$\begin{aligned}\hat{h} &= W_0x + \Delta\hat{W}x + \Delta W_k x \\ &= W_0x + \hat{B}\hat{A}x + B_kA_kx \\ &= W_0x + (\sum_{i=1}^{n}\omega_iA_i)(\sum_{i=1}^{n}\omega_iB_i)x + B_kA_kx.\end{aligned} \tag{5}$$

where A_k and B_k represent the additional LoRA weights that require updates through fine-tuning. This approach allows LLMs to leverage the knowledge and ability encoded in trained LoRA plugins from various domains, enhancing their performance on the target domain database.

Figure 10 illustrates the process of the weights merging-based few-shot LoRA-based fine-tuning method. It first extracts LoRA modules from Plugin Hub and merges them. Then, the multi-task dataset generated from Section 6.1 is used for fine-tuning the LoRA model. After fine-tuning, the trained LoRA module is saved to the Plugin Hub.

8 OUTPUT CALIBRATION

Due to the hallucination and decoding strategy of LLMs, LLM-based text-to-SQL models often generate invalid and syntactically incorrect SQL queries. For instance, there may be non-existent table and column names, and incorrectly associated relationships between tables and columns in the generated SQL queries. There are some examples of the generated invalid SQL queries, as shown in Figure 11.

To calibrate the output of LLMs and enhance its correctness, this paper proposes an efficient output calibration algorithm. The proposed method calibrates the outputs without executing the SQL queries, which addresses the execution-dependency issue that occurred in the state-of-the-art methods (e.g., NatSQL [7, 19], execution-based self-consistency [6, 31]) and can be directly used in the real financial applications. The details of this method are presented in Algorithm 1.

The input of the algorithm comprises a list of n candidate SQL queries generated in parallel by the LLMs, along with the corresponding schema information for these SQL queries. The objective

Algorithm 1: Output Calibration Step

Input: the list of n candidate SQL queries Q;
the schema information of the SQL queries S
Output: the final SQL query $\hat{q}$

```
initialize E = Dict{}, which maps SQL to SQL keyword components;
foreach q_i ∈ Q do
    q_i ← f_1(q_i, S)  // fix some typo errors;
    e_i ← f_2(q_i, S)  // extract keyword components;
    if columns of e_i in S then
        E[q_i] = e_i
initialize C = List(), the cluster list of the SQL queries;
foreach q_i, e_i ∈ E do
    foreach C_j ∈ C do
        q_j = C_j[0], e_j = E[q_j]
        if e_i compatible with e_j then
            C_j.append(q_i)
            break
    if q_i doesn't match any cluster in C then
        C.append([q_i])
sort all C_i ∈ C in descending order according to len(C_i);
q̂ ← C_1[0]
q̂ ← f_3(q̂, S)  // align tables to columns
return q̂
```

of our algorithm is to identify the most consistent and valid SQL query.

Due to the inherent randomness of LLMs, the SQL queries they generate frequently contain grammatical errors, which hinders the process of SQL parsing. Thus, we rectify some typo errors for each SQL query, such as replacing " == " with " = " or addressing issues where the "JOIN ON" keyword is used without specifying the corresponding foreign key. When encountering an invalid column, we employ a fuzzy matching approach to replace it with the column from the schema that is most similar in terms of characters. These problems would affect the subsequent extraction of SQL keywords and their values. Subsequently, we parse the modified SQL and extract the keywords with their corresponding values, which are used to apply a non-execution-based self-consistency method. Specifically, we determine the equivalence of two SQL queries by assessing the consistency of their SQL keywords and values. Using this criterion, we cluster equivalent SQL queries together. Then, we select one SQL query from the largest cluster as the result. This approach significantly enhances the consistency of SQL queries.

Finally, we verify whether the SQL query associates tables with columns and then align the tables with their respective columns. For every column specified in the SQL, this module guarantees that the corresponding table must be included in the *FROM* clause. In instances where *table.column* or *alias.column* is encountered, the module verifies whether the specified *column* is indeed associated with the specified *table* or the *alias table*. If not, the module searches for the appropriate *table* containing the specified *column* within the *FROM* clause. If the table is not found through this search, the module selects one table from the schema information.

Traditional SQL calibrating methods involve executing the SQL queries within the databases to identify and rectify errors, which enhances the overall quality of the SQL. However, such approaches are not applicable to financial analysis scenarios, where the databases are often huge, and executing SQL queries can be time-consuming. In contrast, our output calibration algorithm relies on the inherent structure and syntactical rules of SQL, efficiently mitigating the impact of illusions and uncertainty within LLMs when calibrating SQL queries. Through such calibration of SQL queries, our approach significantly improves the consistency and utility of the final SQL queries.

9 EXPERIMENTS

In this section, we perform experimental studies to validate the effectiveness of FinSQL in financial Text-to-SQL. We first compare the overall performance of GPT-based methods and T5-based methods. Subsequently, we display the effectiveness of our parallel Cross-Encoder model. Then we perform ablation studies on our Hybrid Data Augmentation and Output Calibration methods. Additionally, we conduct substantial experiments to validate the excellent performance of our weights merging methods in few-shot cross-database model transfer.

9.1 Experiment Setup

Datasets. We conduct experiments on our financial Text-to-SQL dataset BULL, which is collected from real industrial scenarios. This dataset comprises three databases, across three common financial domains: fund, stock, and macro economy. On average, each database contains 26 tables and 390 columns. It includes 4966 question-SQL query pairs, with 1744 training examples and 405 development examples for fund, 1672 training examples and 464 development examples for stock, and 550 training examples and 131 development examples for macro economy. Notably, The dataset is available in both English (denoted as BULL-en) and Chinese (denoted as BULL-cn) versions.

Evaluation Metrics. We choose execution accuracy (EX) as our evaluation metric, as implemented by Test Suite Accuracy [42]. This metric is also the official evaluation metric used by the popular Text-to-SQL leaderboard, Spider [39]. EX executes the predicted SQL query and golden SQL query in the database and judges whether the two have the same execution results.

Implementation. We employ both decoder-only architecture models, LLaMA2 [32] and Baichuan2 [37], as well as encoder-decoder architecture models, T5 [27] and mT5 [36], as the base large language models for fine-tuning. LLaMA2 comprises a range of large language models with a parameter scale spanning from 7B to 70B. LLaMA is currently one of the most effective and influential open-source large language models. It holds a leading position in many English benchmarks compared to other open-source models. Therefore, we choose LLaMA2 as the base model and fine-tune it on BULL-en. Baichuan2 is a collection of large multilingual language models, encompassing 7B and 13B parameters, trained from scratch on a corpus of 2.6 trillion tokens. Baichuan2 demonstrates remarkable performance on well-established benchmarks. Considering its proficiency in the Chinese language, we fine-tune it on BULL-cn. T5, an encoder-decoder model, is pre-trained on an amount of unsupervised and supervised multi-tasks. Additionally, mT5 is a multilingual variant of T5 that covers 101 languages. Methods based on T5 achieve SOTA on the Sider leaderboard of fine-tuning setting. Therefore, we also choose T5 and mT5 as the base models to handle English and Chinese tasks, respectively. We employ Cross-Encoder models based on Roberta-large [24] and Chinese-Bert-large [3] for BULL-en and BULL-cn, respectively. All the fine-tuning experiments are conducted with LoRA on a single A40 GPU.

Baseline. We conduct experiments on our financial Dataset and compare it with the following baselines: (1) **DAIL-SQL** [8] improves the selection process by encoding structured knowledge as SQL statements, selecting examples based on skeleton similarities, and removing cross-domain knowledge from examples for token efficiency. (2) **DIN-SQL** [25] enhances the performance of LLM-based text-to-SQL models by implementing a strategic task decomposition approach. It also brought in adaptive prompting techniques that are uniquely adjusted to the complexity of specific tasks. (3) **C3** [6] integrates Clear Prompting, Calibration with Hints, and Consistent Output, significantly enhancing accuracy and systematizing processes in GPT-based Text-to-SQL tasks. (4) **RESDSQL** [19] proposes a ranking-enhanced encoding and skeleton-aware decoding framework, which utilizes a two-stage method to retrieve relevant schema items first and then generate the SQL queries based on T5. It is the best fine-tuning based method on Spider leaderboard; (5) **Token Preprocessing** [28] inserts spaces to separate words within schema and question tokens, improving their readability and semantic clarity. (6) **Picard** [30] employs incremental parsing to restrict auto-regressive decoders in language models, effectively filtering out unsuitable tokens to improve the precision of text-to-SQL translations.

9.2 Overall Performance

In Table 4 and Table 5, we report the performance of our FinSQL method against other leading baseline methods from Spider leaderboard on our proposed dataset of English and Chinese versions respectively. On the BULL-en, we combine FinSQL with LLaMA2 and T5 to compare with GPT-based methods and T5-based methods respectively. On the BULL-cn, we employ Baichuan2 and mT5 to better adapt to the Chinese context. Our FinSQL method outperforms all other approaches on both BULL-en and BULL-cn. On BULL-en, FinSQL exhibits an EX of 82.2% with LLaMA and 81.5% with T5. On BULL-cn, FinSQL displays an EX of 76.6% with Baichuan2 and 70.4% with mT5.

GPT-based methods. For methods utilizing GPT-4 and ChatGPT, we calculate the Cost Per SQL based on the unit price in Table 2 and the number of input and output tokens. Due to budget constraints, we only select 20 entries for GPT-4 based methods and 100 entries for ChatGPT-based methods. For DIN-SQL + GPT-4, the total length of the prompt exceeds GPT-4's token limit of 8192, rendering the test unfeasible. Although GPT-4-32k can meet our needs of context length, the price is too high for us to afford, hence we only estimate the api cost. The methods based on GPT show varied results: DAIL-SQL + GPT-4 achieves a considerable EX of 75%, which is around 7% lower than FinSQL + LLaMA2 and 1.6% lower than FinSQL + Baichuan2. On BULL-en, The EX of DAIL-SQL + ChatGPT and C3 + ChatGPT is 46% and 7% respectively. On BULL-cn, the numbers are 55% and 2%, which are not comparable with

Table 4: Overall results of different previous methods on BULL-en. For fine-tuning methods, we employ T5-large and LLaMA2-13B. The * means we employ our parallel Cross-Encoder model for schema linking.

Model	EX	Cost Per SQL($)
DIN-SQL [25] + GPT-4	-	4.9103
DAIL-SQL [8] + GPT-4	75.0	0.1579
DAIL-SQL [8] + ChatGPT	46.0	0.0051
C3[6] + ChatGPT	7.0	0.0065
RESDSQL* [19] + T5	78.8	-
Token Preprocessing* [28] + T5	67.5	-
Picard* [30] + T5	79.3	-
FinSQL + LLaMA2	82.2	-
FinSQL + T5	81.5	-

Table 5: Overall results of different previous methods on our BULL-cn. For fine-tuning methods, we employ mT5-large and Baichuan2-13B. The * means we employ our parallel Cross-Encoder model for schema linking.

Model	EX	Cost Per SQL($)
DIN-SQL [25] + GPT-4	-	4.9158
DAIL-SQL [8] + GPT-4	75.0	0.1581
DAIL-SQL [8]+ ChatGPT	55.0	0.0053
C3 [6] + ChatGPT	2.0	0.0078
RESDSQL* [19] + mT5	66.9	-
Token Preprocessing* [28] + mT5	60.2	-
Picard* [30] + mT5	72.7	-
FinSQL + Baichuan2	76.6	-
FinSQL + mT5	70.4	-

the other methods. We speculate that this is due to their specific prompts for Spider dataset, which results in exhibiting poor performance when facing data variability. In addition to incurring extra costs, GPT-based methods require several hours to infer results from less than 100 data entries. In contrast, FinSQL is better suited to adapt to real-world data scenarios, thus holding high practical value and cost-effectiveness.

T5-based methods. Due to context limitations, T5-based methods are unable to process our concatenated input sequences. Therefore, we initially utilize our parallel Cross-Encoder for retrieval, simplifying each question's schema to 3 tables and 7 columns. As a result, T5-based methods achieve a higher accuracy compared to GPT-based methods. they are all able to achieve an execution accuracy rate of over 60%. It is worth noting that Picard demonstrates a substantial accuracy rate, making it the best-performing method after FinSQL. We surmise that this is due to its additional filtering of irrelevant tokens. Besides, as for methods applied to mT5, FinSQL performs 2.3% lower than Picard. Although FinSQL still has shortcomings in BULL-ch, it has provided valuable insights into the challenges and complexities of Text-to-SQL in financial analysis, guiding our future research directions.

Table 6: Performance of Schema Linking

Schema Item	Table	Column
AUC (BULL-en)	0.9984	0.9979
AUC (BULL-cn)	0.9995	0.9994

Table 7: The recall@k of tables and columns of our Parallel Cross-Encoder model.

Dataset	Table			Column		
	R@3	R@5	R@10	R@5	R@7	R@10
BULL-en	99.1	99.5	99.8	96.6	98.0	99.0
BULL-cn	99.5	99.9	100.0	97.1	98.6	99.5

Table 8: Effective of different data augmentation methods.

Technique	EX (English)	EX (Chinese)
Hybrid Data Augmentation	80.7	75.3
w/o CoT Data	78.3 (-2.4)	73.4 (-1.9)
w/o Synonyms Data	77.0 (-2.7)	73.8 (-1.5)
w/o Skeleton Data	78.1 (-2.6)	71.0 (-4.3)
w/o Augmented Data	76.9 (-3.8)	70.6 (-4.7)

9.3 Effect of Schema Linking

To evaluate the efficacy of our schema linking method, we adopt the Area Under the ROC Curve (AUC) as our metric. As presented in Table 6, the outcomes reveal an AUC of 0.9984 for tables and 0.9979 for columns in the BULL-en, alongside 0.9995 for tables and 0.9994 for columns in the BULL-cn. These metrics validate the robustness of our approach, demonstrating the precise identification of relevant schema items in both datasets.

Additionally, we conduct experiments on our parallel cross-encoder model, with detailed results in Table 7. The model's effectiveness is measured using the recall at k (R@k) metric, which evaluates its ability to correctly identify relevant items within the top k selections. In the BULL-en, the model exhibite outstanding results for table recall, achieving 99.1%, 99.5%, and 99.8% for R@3, R@5, and R@10 respectively. In BULL-cn, it shows superior results with scores of 99.5%, 99.9%, and a perfect 100.0%. In the more complex task of column recall, the model sustains strong performance with 96.6%, 98.0%, and 99.0% for R@5, R@7, and R@10 in BULL-en, and 97.1%, 98.6%, and 99.5% in BULL-cn. These results highlight the model's proficiency in managing multilingual datasets with remarkable precision and recall, a crucial aspect for the subsequent stages of Text-to-SQL task.

9.4 Effect of Data Augmentation

In this section, we perform an ablation study to examine the impact of various data augmentation techniques. To eliminate influence from other unrelated factors and focus solely on the impact of data augmentation, we did not perform output calibration in this study. The experimental results are illustrated in Table 8. It is evident that the exclusion of each specific augmentation method degrades the performance. When CoT Data is omitted, the execution accuracy

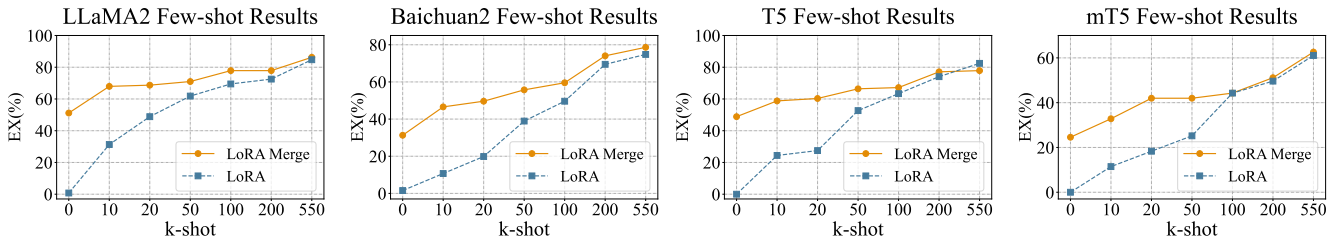

Figure 12: Execution accuracy of weights merging-based few-shot LoRA on four models.

experiences a reduction of 1.9% for Chinese and 2.4% for English, confirming the significance of this particular data enhancement. Similarly, the absence of Synonyms Data leads to a decrease of 1.5% for Chinese and 2.7% for English, while the removal of Skeleton Data results in a decline of 4.3% for Chinese and 2.6% for English. The most notable impact is observed when all forms of Augmented Data are excluded, culminating in a decrease of 4.7% for Chinese and 3.8% for English. These results collectively validate the necessity of the hybrid data augmentation approach in preserving the robustness of FinSQL.

9.5 Effect of Weights Merging-based Few-shot LoRA

In this section, we discuss the role of our Weights Merging-based Few-shot LoRA in a low-sample scenario. In our dataset, there are 1744 and 1672 training samples for fund and stock respectively, while macro economy only has 550 data samples. Consequently, we separately train LoRA models on fund and stock datasets, and subsequently merge these two LoRA modules using the average weighting. Following the merging process, we continue fine-tuning the merged LoRA model with varying amounts of data in macro economy. We also train the Cross-Encoder model with the combination data of funds, stocks, and varying amounts of macro data to retrieve relevant schema items. The results are shown in Figure 12.

From the Figure, we can observe that whether it is LLaMA2, Baichuan2, T5, or mT5, the performance using LoRA Merge is usually higher than LoRA, with a greater disparity in performance at lower k-shot values. Across the four models, for the zero-shot learning setting, the performance of LoRA-Merge surpasses that of LoRA by 50.39% in LLaMA2, 29.77% in Baichuan2, 47.33% in T5, and 24.57% in mT5. For 10-shot learning, LoRA-Merge's performance exceeds LoRA by 36.64%, 35.87%, 34.35%, and 21.37% respectively. As the number of few-shot instances increases, the disparity gradually diminishes. When the shot number exceeds 200, the trained LoRA sometimes performs better than the merged LoRA. However, we can still observe the significant benefits of merging LoRA weights. When there is a scarcity of training data, the pre-merged LoRA module fine-tuned on other tasks exhibits a more pronounced performance than LoRA module trained from scratch. Hence this method is particularly suitable for low-sample scenarios.

9.6 Effect of Output Calibration

To investigate the impact of output calibration techniques on model performance, we conduct an ablation study as depicted in Table 9.

Table 9: Effect of Output Calibration

Technique	EX (Chinese)	EX (English)
FinSQL	76.6	82.2
w/o Output Calibration	75.3 (-1.3)	80.7 (-1.5)
w/o Self-Consistency	76.4 (-0.2)	81.9 (-0.3)
w/o Alignment	75.5 (-1.1)	81.0 (-1.2)

The output calibration is comprised of two methodologies: self-consistency and alignment. The results are quantified in terms of execution accuracy for Chinese and English.

Our findings indicate that the exclusion of output calibration results in a reduction of EX accuracy by 1.3% for Chinese and 1.5% for English. The absence of self-consistency leads to a downtrend of 0.2% for Chinese and 0.3% for English. Lastly, dropping alignment leads to a decrease of 1.1% in EX accuracy for Chinese and 1.2% for English. As expected, the cumulative impact of removing self-consistency and alignment equates to the elimination of output calibration. This further confirms that the observed decrease in EX is indeed related to the elimination of output calibration in both linguistic contexts.

10 CONCLUTION

In this paper, we propose BULL, a Text-to-SQL dataset derived from real-world financial scenarios. Based on this dataset, we propose FinSQL, a model-agnostic LLMs-based Text-to-SQL framework for financial analysis. FinSQL conducts hybrid data augmentation and parallel Cross-Encoder to construct concise and diversified prompts. Besides, FinSQL utilizes LoRA merging methods to improve the performance for cross-database model transfer. Moreover, FinSQL proposes an output calibration method to improve the quality of SQL queries generated by LLMs. Extensive experiments on BULL demonstrate the effectiveness of FinSQL in financial analysis. The BULL and FinSQL have the potential to forge a new trend in financial Text-to-SQL research.

ACKNOWLEDGMENTS

This work was supported in part by the NSFC under Grants No. (62025206, U23A20296, and 62302436), Zhejiang Province's "Lingyan" R&D Project under Grant No 2024C01259, Ningbo Science and Technology Special Projects under Grant No. 2023Z212, Yongjiang Talent Programme "Research on key technologies in Intelligent Database". Yuren Mao is the corresponding author of this work.

REFERENCES

[1] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations. In *ACL*.

[2] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. PaLM: Scaling Language Modeling with Pathways. *Journal Of Machine Learning Research* 24 (2023), 240:1–240:113.

[3] Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Shijin Wang, and Guoping Hu. 2020. Revisiting Pre-Trained Models for Chinese Natural Language Processing. In *EMNLP Findings*.

[4] H Dai, Z Liu, W Liao, X Huang, Y Cao, Z Wu, L Zhao, S Xu, W Liu, N Liu, et al. 2023. AugGPT: Leveraging ChatGPT for Text Data Augmentation. *arXiv preprint arXiv:2302.13007* (2023).

[5] Naihao Deng, Yulong Chen, and Yue Zhang. 2022. Recent Advances in Text-to-SQL: A Survey of What We Have and What We Expect. In *COLING*.

[6] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. *arXiv preprint arXiv:2307.07306* (2023).

[7] Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John Drake, and Qiaofu Zhang. 2021. Natural SQL: Making SQL Easier to Infer from Natural Language Specifications. In *EMNLP*.

[8] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363* (2023).

[9] Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Sam Madden, and Xiaoyong Du. 2023. Few-shot Text-to-SQL Translation using Structure and Content Prompt Learning. In *SIGMOD*.

[10] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *ACL*.

[11] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. In *ICML*.

[12] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *ICLR*.

[13] Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Yanyang Li, Bowen Li, Jian Sun, and Yongbin Li. 2022. S^2SQL: Injecting Syntax to Question-Schema Interaction Graph Encoder for Text-to-SQL Parsers. In *ACL*.

[14] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. In *ACL*.

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

[16] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic Evaluation of Text-to-SQL Parsers. In *ACL*.

[17] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the Role of Schema Linking in Text-to-SQL. In *EMNLP*.

[18] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *EMNLP*.

[19] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. In *AAAI*.

[20] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023. Graphix-T5: Mixing Pre-trained Transformers with Graph-Aware Layers for Text-to-SQL Parsing. In *AAAI*.

[21] Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *NeurIPS*.

[22] Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *ACL*.

[23] Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Teaching Models to Express Their Uncertainty in Words. *Transactions on Machine Learning Research* (2022).

[24] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019).

[25] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-sql with self-correction. In *NeurIPS*.

[26] Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, et al. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629* (2022).

[27] 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. *Jounal Of Machine Learning Research* 21 (2020), 140:1–140:67.

[28] Daking Rai, Bailin Wang, Yilun Zhou, and Ziyu Yao. 2023. Improving Generalization in Language Model-based Text-to-SQL Semantic Parsing: Two Simple Semantic Boundary-based Techniques. In *ACL*.

[29] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP*.

[30] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *EMNLP*.

[31] Ruoxi Sun, Sercan O Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023. SQL-PaLM: Improved Large Language ModelAdaptation for Text-to-SQL. *arXiv preprint arXiv:2306.00739* (2023).

[32] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[33] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *ACL*.

[34] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*.

[35] Dingjun Wu, Jing Zhang, and Xinmei Huang. 2023. Chain of Thought Prompting Elicits Knowledge Augmentation. In *ACL*.

[36] Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer. In *NAACL*.

[37] Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, Fan Yang, et al. 2023. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305* (2023).

[38] Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018. Syntaxsqlnet: Syntax tree networks for complex and cross-domain text-to-sql task. *arXiv preprint arXiv:1810.05237* (2018).

[39] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP*.

[40] John M. Zelle and Raymond J. Mooney. 1996. Learning to Parse Database Queries Using Inductive Logic Programming. In *AAAI*.

[41] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

[42] Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic Evaluation for Text-to-SQL with Distilled Test Suites. In *EMNLP*.

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