FATO-SQL: a comprehensive framework for high-performance Text-to-SQL task

Yongnan Chen*, Shijia Gu, Zixiang He
Kunlun Digital Technology Co., Ltd., Beijing 102206, China

ABSTRACT

Text-to-SQL tasks aim to bridge the gap between natural language questions and SQL queries, enabling efficient interaction with databases without the need for expertise in SQL coding. In this paper, we introduce FATO-SQL, a novel Large Language Model (LLM)-based framework designed for medium-scale LLMs to generate complex SQL queries. FATO-SQL leverages the Retrieval-Augmented Generation (RAG), prompting engineering, and two rounds of LLM calls for SQL generation and diverse response generation. We implemented FATO-SQL using production data from the petrochemical industry, testing it with multiple tables joins and multi-level nested SQL queries. Results show that FATO-SQL achieves an overall accuracy of 94% on 40 testing questions. The FATO-SQL framework demonstrates promising potential for practical industrial applications, highlighting its efficacy and adaptability in real-world scenarios.

Keywords: Large language model, Text-to-SQL, agent, RAG, prompt engineer, petroleum intelligence

1. INTRODUCTION

The Text-to-SQL task aims to automatically generate Structured Query Language (SQL) queries in response to corresponding natural language (NL) questions with the assistance of models and programs[1]. This long-standing challenge is crucial for enhancing database accessibility without necessitating expertise in SQL coding. With the rapid advancement of Large Language Models (LLMs), solutions for the Text-to-SQL task have evolved from relying on rule-based methods to incorporating LLMs.

In the early stages, Text-to-SQL research focused on rule-based methods[2]. Given that SQL queries follow a strongly formatted language, designing templates to match user questions was an efficient approach. Subsequently, research shifted towards using sequence-to-sequence frameworks to address the Text-to-SQL task. Notable encoding methods include table-aware approaches[3,4], anonymous encoding[5], graph neural networks (GNNs)[6], and the BERT-based model Tabert[7]. Decoding methods encompass reinforcement learning[8], sketch-based approaches and multitasking[9], and abstract syntax networks[10]. Recent advancements have highlighted the impressive performance of Large Language Models (LLMs) in this domain[11-14]. The effectiveness of the LLMs is largely attributed to their emergent abilities[15,16] and strong reasoning capabilities. Current research on LLM-based Text-to-SQL[17-23] primarily explores in-context learning strategies and supervised fine-tuning with data specific to the target domain.

However, this task faces two main challenges. First, the current accuracy on the Spider dataset[24], with 91% achieved by DAIL-SQL[25], falls short of the requirements for industrial applications and implementations. The SQL generation accuracy needs to be as high as possible for practical decision-making level services. Second, the scale of LLM parameters and the resulting computational cost are often constrained by available resources. Therefore, in practical applications, large language models with fewer than 13 billion parameters (referred to as medium-scale LLMs) are commonly used[26]. This presents a significant challenge: how to maximize the potential of medium-scale LLMs for the Text-to-SQL task and achieve satisfactory SQL generation accuracy.

To address this challenge, we propose a comprehensive framework called FATO-SQL (Four-stage Actions with Two Rounds Call of LLM) to assist medium-scale LLMs in completing the Text-to-SQL task. The framework includes the Retrieval-Augmented Generation (RAG) process, which starts with the first call to the LLM for SQL generation, followed by SQL execution, and concludes with the second call to the LLM for NL response generation. By dividing the task into two sub-tasks through two rounds of LLM calls, medium-scale LLMs can concentrate on specific objectives

*chenyongnan@cnpc.com.cn

each time. Additionally, the prompts for each call can incorporate precise and comprehensive contextual information, facilitating the generation of accurate SQL queries and responses.

Contributions. Our contributions to the Text-to-SQL task can be summarized as follows:

(1) We introduce a framework consisting of four stages, enabling medium-scale LLMs to infer complex SQL queries.

(2) The framework incorporates two rounds of LLM calls, thereby enhancing the completeness and accuracy of the Text-to-SQL task.

(3) We define structures for prompting and vector databases to facilitate a more accurate understanding of user intent.

2. TASK STATEMENT

The implementation of Text-to-SQL task involves answering a NL question *q* on a target database *D*. Let *T* be the tables in *D*, *C* be the columns of *T*, and *F* be the foreign key relations. The database schema *S* of *D* can be defined as *S*={*T*, *C*, *F*}, where $T \in \{T_1, T_2, ..., T_m\}$, $C \in \{C_1^1, C_2^1, ..., C_{n_1}^1, ..., C_1^m, C_2^m, ..., C_{n_m}^m\}$ and $F \in \{(C_h^i, C_k^j)\}$. Let *s* be the SQL query that can response the question *q*. The Text-to-SQL task can therefore be formulated as:

$$s = f(q, \{T, C, F\} \mid \theta) \tag{1}$$

where the function $f(\sim \mid \theta)$ represent a LLM model with parameter θ.

3. METHOD

3.1 Overview

In this work, we introduce a four-stage collaborative Text-to-SQL framework (Figure 1), where the LLMs are adopted as intelligent agent that assist throughout the framework for SQL generation and execution. The entire framework consists of two rounds of LLM calls, with a RAG process to aid in prompt generation, and an external step to execute the SQL query generated by LLM. The first call accomplishes the task of SQL generation, while the second call completes the analysis and representation of the SQL results. Algorithm 1 outlines the relationships of these components.

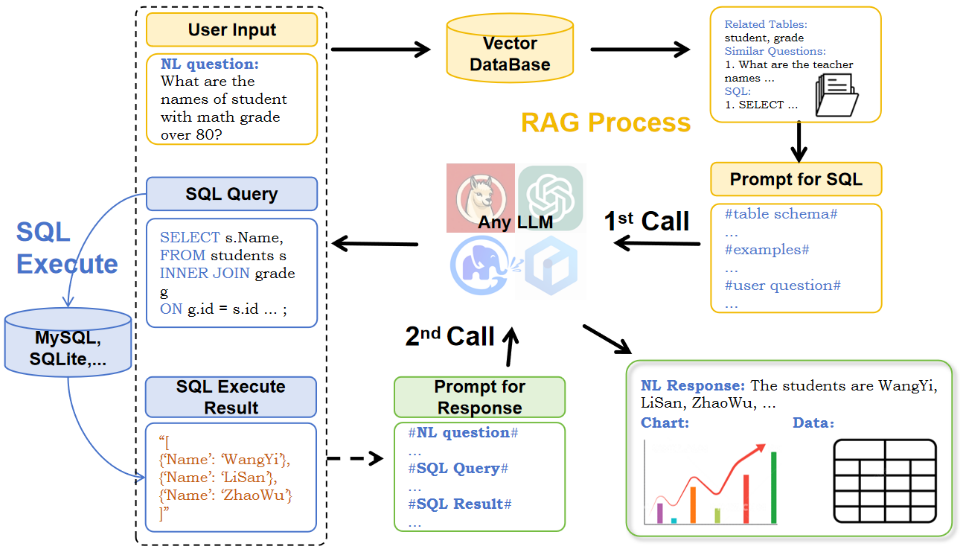

User Input
NL question: What are the names of student with math grade over 80?
Vector DataBase
Related Tables: student, grade
Similar Questions: 1. What are the teacher names ...
SQL: 1. SELECT ...
RAG Process
Prompt for SQL
#table schema# ... #examples# ... #user question# ...
1st Call
Any LLM
SQL Query
SELECT s.Name, FROM students s INNER JOIN grade g ON g.id = s.id ... ;
SQL Execute
MySQL, SQLite,...
SQL Execute Result
"[{'Name': 'WangYi'}, {'Name': 'LiSan'}, {'Name': 'ZhaoWu'}]"
2nd Call
Prompt for Response
#NL question# ... #SQL Query# ... #SQL Result# ...
NL Response: The students are WangYi, LiSan, ZhaoWu, ...
Chart:
Data:

Figure 1. Four-stage Text-to-SQL implementation framework.

Note: Stage 1: RAG process. The RAG process retrievals the relative tables schema and SQL queries from the vector database. Stage 2: First call of LLM. The relative information is used for building a prompt for LLM. The LLM is forced to return a SQL query to answer the NL question. Stage 3: SQL Execute. The SQL Query is executed in the database and returns the resulting data. Stage 4: Second call of LLM. The prompt is consisting of the initial NL question, the SQL query, and the SQL execute results. The LLM generates a comprehensive response that includes an NL response, the resulting data table and the chart format to visualize the data.

Algorithm 1 The algorithm of FATO-SQL framework

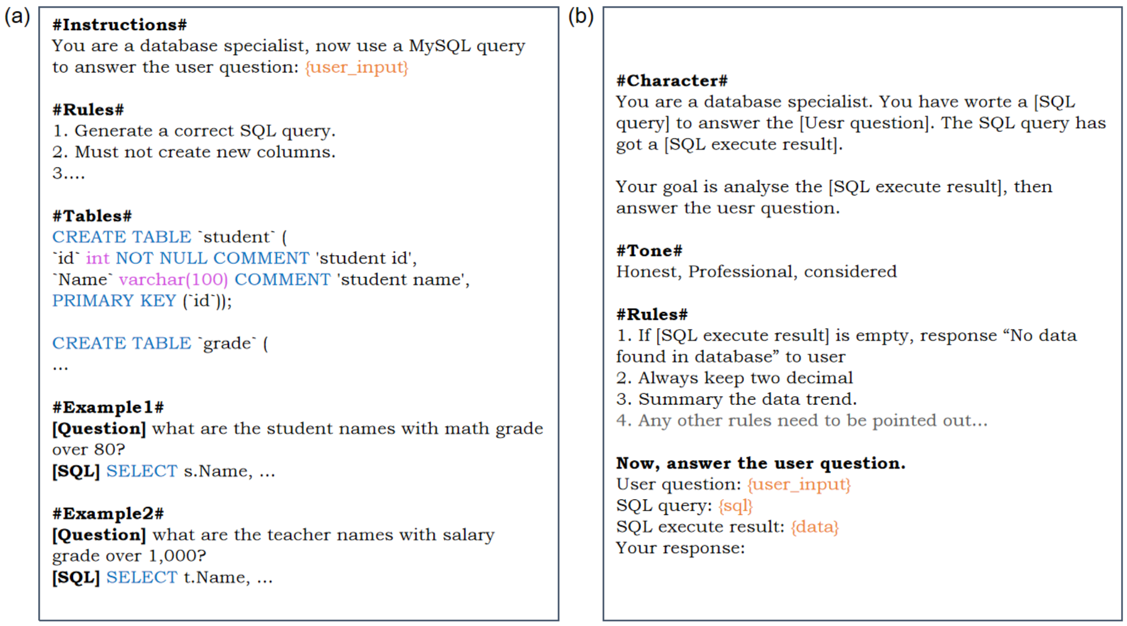
```
Input: User question q, Vector Database vd, SQL database sd
Output: SQL query sql, NL response r, Chart c
# STAGE 1: RAG process #
 terms = [Similarity(q, $vd_1$)]    # identify the professional terms in user question
 T = Find_table_schema(terms)
 q' = replace special words with terms # process users' question
 q'' = [Similarity(q', $vd_2$)]                    # the similar NL questions
 if cos_distance(q',q'') < threshold do
     prompt = Generate_prompt_few_shots(q', T, q'')
 else
     prompt = Generate_prompt_zero_shot(q', T)
 end if
#STAGE 2: $1^{st}$ time call to generate SQL query #
sql = CallLLM$1^{st}$(prompt)
# STAGE 3: SQL execute #
if ExecuteSQL(sql, sd) do not raise error do
    data = ExecuteSQL(sql, sd)
    return r, c = CallLLM$2^{nd}$(q', s, data) # STAGE 4: $2^{nd}$ time call to generate response #
else
    return r = CallLLM$2^{nd}$(q', s, ErrorMessage)
end if
```

3.2 Prompting for SQL generation

Prompt engineering is a crucial aspect of the Text-to-SQL task, and various prompting templates have been proposed to facilitate accurate SQL generation. Representative prompting templates include the OPAI Demonstration Prompt[27], Code Representative Prompt[28,29], Alpaca SFT Prompt[30], and DAIL Organization Prompt[25]. These prompting structures share several common components, including generation instructions, database schema, and examples such as well-written SQL queries paired with their corresponding NL questions. The differences between them mainly lie in the formats of these components. For example, the database schema can be structured as a table name with a list of columns or represented with the Data Definition Language (DDL) of the tables.

Limited by the reasoning capabilities of medium-scale LLMs, the prompting strategy in our framework is designed to be as informative and precise as possible. Four main components are included: Instructions, Rules, Tables, and Examples, as shown in Figure 2a. The Instructions component sets the characteristics and targets for the LLM. The Rules part emphasizes the SQL generation requirements for the LLM. The Tables part retrieves the Data Definition Language (DDL) of the tables related to the specific user question from the SQL database. And, the Examples section lists questions similar to the user input question and their corresponding correct SQL queries.

The Instruction and Rules sections can be flexibly adapted to the needs of vertical fields. For example, the "ORDER BY" statement is necessary in a ranking scenario but can easily cause execution errors in a calculation scenario. Such statements, including "ALIAS," "DATE," etc., often have different usage requirements in different areas. Therefore, in addition to the general instructions listed in Figure 2a, these specific requirements can also be included in the Instruction and Rules sections.



(a)

```
#Instructions#
You are a database specialist, now use a MySQL query
to answer the user question: {user_input}

#Rules#
1. Generate a correct SQL query.
2. Must not create new columns.
3....

#Tables#
CREATE TABLE `student` (
`id` int NOT NULL COMMENT 'student id',
`Name` varchar(100) COMMENT 'student name',
PRIMARY KEY (`id`));

CREATE TABLE `grade` (
...

#Example1#
[Question] what are the student names with math grade
over 80?
[SQL] SELECT s.Name, ...

#Example2#
[Question] what are the teacher names with salary
grade over 1,000?
[SQL] SELECT t.Name, ...
```

(b)

```
#Character#
You are a database specialist. You have worte a [SQL
query] to answer the [Uesr question]. The SQL query has
got a [SQL execute result].

Your goal is analyse the [SQL execute result], then
answer the uesr question.

#Tone#
Honest, Professional, considered

#Rules#
1. If [SQL execute result] is empty, response "No data
found in database" to user
2. Always keep two decimal
3. Summary the data trend.
4. Any other rules need to be pointed out...

Now, answer the user question.
User question: {user_input}
SQL query: {sql}
SQL execute result: {data}
Your response:
```

Figure 2. Example of promptings. (a) represents the prompting structure for SQL query generation; (b) represents the prompting structure for response generation. The variable names that needed to be formatted are enclosed with curly braces. The specific details are omitted.

3.3 RAG process

Regarding the third and fourth parts, question-related tables and similar examples, the Retrieval-Augmented Generation (RAG) process is involved. RAG is a technique that combines retrieval-based and generation-based methods. In a RAG framework, the LLM retrieves relevant documents or passages from a large corpus of text and uses this retrieved information to generate more accurate and contextually appropriate responses[31]. Therefore, RAG is particularly useful in tasks requiring detailed knowledge and precise answers, such as Text-to-SQL.

To implement the framework, at least two vector databases are required. One is to store the terminologies that appear in the SQL database, while the other is to store pairs of NL questions and SQL query answers. Embedding indices are built for the terminologies and the NL questions respectively.

Once the system receives a user question and creates the embedding vector for this question, it searches all the terminologies to find the closest terminology and consequently identifies the tables related to this terminology. The Data Definition Language (DDL) information of the selected tables can then be retrieved from the SQL database. Subsequently, the system searches the NL questions to identify a few similar questions and retrieves their corresponding SQL query answers.

It is worth noticing that the design of vector databases, along with the prior question-answer pairs, provides impressive flexibility for extra-hard SQL generation. Especially in particular areas where the terminologies rarely appear in the training data of the pre-trained LLMs, the recognition and classification of these terminologies are crucial for the model to understand the meaning of the user questions.

3.4 The first call of LLM

In the first call of LLM, the previous prompt (Figure 2a) serves as input, and the LLM is tasked with generating a SQL query that can answer the user question. Then, in the third stage of our framework, the generated SQL query is passed into the SQL database to execute and return the results in JSON format. It's important to note that if the generated SQL query has a grammar error, the execution program returns an error message for the next step.

3.5 The second call of LLM

Results representation for SQL execution outputs are expected to be comprehensive. Several platforms[32-34] have been developed to present outputs in a structured manner, typically including the SQL query and thoughts about the query, a table displaying the SQL execution results, and visualizations such as line charts or bar charts. However, the demand for accurate SQL query generation alongside additional analysis and judgment tasks can overwhelm the reasoning ability of medium scale LLMs, potentially resulting in low-quality responses.

Therefore, in this work, the LLM is called twice. The first call serves as a SQL generation and execution agent, as illustrated in Section 3.4. After obtaining the SQL execution results, the second call of LLM acts as a data analysis assistant, automatically refining this information and enhancing the response.

Figure 2b illustrates the prompting example for the second call. The structure includes Character, Tone, Rules, and Context. The overall purpose of the second call is to analyze the SQL execution data and answer the user question. Therefore, the character of the LLM is defined as a helpful assistant, and specific rules for generating the response are listed, such as the units of the data and the decimal places. The contextual information, including the user question, SQL query, and SQL execution results, should also be listed in the prompt. Additionally, more functions can be added to the second call, such as allowing the model to decide which chart is more suitable for representing the results data.

Through the implementation of the two-round call of LLM framework, the entire Text-to-SQL task is divided into two sub-tasks. As a result, the framework gains the capacity of a large-scale model while utilizing only medium-scale LLMs. However, with the introduction of the second call, the LLM has the opportunity to comprehensively summarize the user question, SQL query, and SQL answer, thereby providing overall feedback about the question. This functionality represents a significant advancement compared to previous platforms.

4. CASE STUDY

4.1 Petrochemical data collection

We implemented the FATO-SQL framework using production data from the petrochemical industry. Six tables were collected, each containing 10 to 20 columns. These tables cover a range of related data, including production equipment information, material information, inventory data, and more. Additionally, we collected 14 frequently used metrics in the petrochemical industry along with their ground truth SQL queries.

4.2 Vector database construction

Compared with leading board datasets such as Spider[35], WikiSQL[8], etc., real production data faces the challenges of complex terminologies, for example, "atmospheric and vacuum distillation unit," and ambiguous column names, for example, "MES_TK_ID" referring to the ID of a tank, and "MTRL_ID" referring to the ID of a material. Pre-trained LLMs struggle to precisely understand such technical terms and table schema.

Therefore, the vector database is designed to cover two aspects. First, the terminology database contains all the special words that appear in the SQL database and the tables they are related to, with embedding vectors assigned to the terminologies. Second, the question-SQL pair database contains regularly used SQL queries in the industry, such as calculations of "yield," "loss rate," and "load rate."

4.3 Test questions

From the perspective of practically use, four types of testing questions are designed, the (1) original questions, the (2) variants of original questions, the (3) random questions related to the database and the (4) questions unrelated to the database. Table 1 lists the representative questions of these four types.

Specifically, the original questions refer to queries that are already loaded in the question-SQL pair database. These queries represent commonly employed metric calculations in the production process, demanding the utmost accuracy due to their significance. The variants of original questions refer to questions asking for the same metrics as the original question, but with altered details such as timestamp, terminology, and sentence order. The prompting for these two types of questions follows the structure shown in Figure 2a.

The random questions related and unrelated to the database consider situations when users have additional needs. For such questions, the RAG process would still map the most related tables and add them to the prompts. Then, the LLM would generate SQL queries accordingly.

Table 1. Case study of the generated SQL queries. Overall accuracy of 40 questions reaches 94%.

Original Question (100%, 14 questions) Calculate the yield of the naphtha output line of Atmospheric and Vacuum Distillation Unit I on April 1, 2023. **Evidence** "naphtha output" is the alias of unit sideline, "Atmospheric and Vacuum Distillation Unit I" is the alias of unit. **Ground Truth** SELECT ROUND(T1.amout_sideline/T2.input*100,2) AS rate FROM (SELECT SUM(balance.amount) AS amout_sideline FROM balance INNER JOIN node ON...INNER JOIN unit ON... WHERE unit.UNIT_ALIAS LIKE '%Atmospheric and Vacuum Distillation Unit I%' AND node.NODE_ALIAS LIKE '%naphtha output%' AND DATE...)AS T1, (SELECT SUM(balance.amount) AS input FROM balance INNER JOIN...WHERE balance.inout_name = 'in' AND...)AS T2; **Predicted (√)** SELECT ROUND(T1.amout_sideline/T2.input*100,2) AS rate FROM (SELECT SUM(balance.amount) AS amout_sideline FROM balance INNER JOIN node ON...INNER JOIN unit ON... WHERE unit.UNIT_ALIAS LIKE '%Atmospheric and Vacuum Distillation Unit I%' AND node.NODE_ALIAS LIKE '%naphtha output%' AND DATE...)AS T1, (SELECT SUM(balance.amount) AS input FROM balance INNER JOIN... WHERE balance.inout_name = 'in' AND...)AS T2;
Variants of Original Questions (100%, 12 question) What is the yield of the atmospheric non-condensable gas feed line of delayed coking unit I on April 1, 2023? **Evidence** "atmospheric non-condensable gas feed" is the alias of unit sideline, "delayed coking unit I" is the alias of unit. **Predicted (√)** SELECT ROUND(T1.amout_sideline/T2.input*100,2) AS rate FROM (SELECT SUM(balance.amount) AS amout_sideline FROM balance INNER JOIN nod ON...INNER JOIN unit ON... WHERE unit.UNIT_ALIAS LIKE '%Delayed Coking Unit I%' AND node.NODE_ALIAS LIKE '%atmospheric non-condensable gas feed%' AND DATE...)AS T1, (SELECT SUM(balance.amount) AS input FROM balance INNER JOIN...WHERE balance.inout_name = 'in' AND...)AS T2;
Questions related to the database (70%, 10 questions) What is the alias of the tank with ID '204'? **Evidence** "204" is the id of tank. **Predicted (√)** SELECT MES_TK_ALIAS FROM tank WHERE MES_TK_ID = 204;
Questions unrelated to the database (without considering accuracy) The maximum altitude reached by the hydrogen hot air balloon in January 2000. **Predicted** (*FATO-SQL generates a SQL query using the available database schema*) SELECT MAX(tank_inventory.tank_con_val) FROM tank_inventory JOIN tank ON.. WHERE tank.MES_TK_ALIAS = 'hydrogen hot air balloon' AND tank_inventory.chk_time BETWEEN '2000-01-01' AND '2000-01-31';

Environments. We conducted all experiments on a server equipped with one NVIDIA V100s (32 GB) GPU and running the Red Hat Enterprise Linux 8.1 (Ootpa) operating system. Llama-7b-v1.5 was utilized for both rounds of LLM calls. The key parameter settings were as follows: temperature=0.1, top_p=0.3, and top_k=30.

5. CONCLUSION

In this work, we propose FATO-SQL, an LLM-based Text-to-SQL framework designed to enable medium-scale LLMs (with fewer than 13 billion parameters) to generate extremely complex SQL queries. The framework comprises the RAG process, prompting structures, two rounds of LLM calls for SQL generation and diverse response generation, and SQL execution in the intermediate steps. We implemented FATO-SQL using production data from the petrochemical industry and tested it with multiple tables joins and multi-level nested SQL queries. The overall accuracy on 40 testing questions reached 94%. In conclusion, FATO-SQL demonstrates its potential for practical industrial applications, showcasing its efficacy and adaptability in real-world scenarios.

REFERENCES

[1] Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Zhang, Q., Yan, Z. and Li, Z., "MAC-SQL: A multi-agent collaborative framework for Text-to-SQL," arXiv abs/2312.11242, (2023).

[2] Popescu, A. M., Etzioni, O. and Kautz, H., "Towards a theory of natural language interfaces to databases," Proceedings of the 8th International Conference on Intelligent User Interfaces, 149-157 (2003).

[3] Sun, Y., Tang, D., Duan, N., et al., "Semantic parsing with syntax-and table-aware SQL generation," arXiv arXiv:1804.08338, (2018).

[4] Hwang, W., Yim, J., Park, S., et al., "A comprehensive exploration on WiKiSQL with table-aware word contextualization," arXiv arXiv:1902.01069, (2019).

[5] Dong, Z., Sun, S., Liu, H., et al., "Data-anonymous encoding for text-to-SQL generation," Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 5405-5414 (2019).

[6] Bogin, B., Gardner, M. and Berant, J., "Global reasoning over database structures for text-to-SQL parsing," arXiv arXiv:1908.11214, (2019).

[7] Yin, P., Neubig, G., Yih, W., et al., "TaBERT: Pretraining for joint understanding of textual and tabular data," arXiv arXiv:2005.08314, (2020).

[8] Zhong, V., Xiong, C. and Socher, R., "Seq2sql: Generating structured queries from natural language using reinforcement learning," arXiv arXiv:1709.00103, (2017).

[9] Xu, X., Liu, C. and Song, D., "Sqlnet: Generating structured queries from natural language without reinforcement learning," arXiv arXiv:1711.04436, (2017).

[10] Yin, P. and Neubig, G., "TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation," arXiv arXiv:1810.02720, (2018).

[11] Ouyang, L., Wu, J., Jiang, X., et al., "Training language models to follow instructions with human feedback," Advances in Neural Information Processing Systems 35, 27730-27744 (2022).

[12] Achiam, J., Adler, S., Agarwal, S., et al., "Gpt-4 technical report," arXiv arXiv:2303.08774, (2023).

[13] Touvron, H., Lavril, T., Izacard, G., et al., "Llama: Open and efficient foundation language models," arXiv arXiv:2302.13971, (2023).

[14] Roziere, B., Gehring, J., Gloeckle, F., et al., "Code llama: Open foundation models for code," arXiv arXiv:2308.12950, (2023).

[15] Singhal, K., Azizi, S., Tu, T., et al., "Large language models encode clinical knowledge," Nature 620(7972), 172-180 (2023).

[16] Brown, T., Mann, B., Ryder, N., et al., "Language models are few-shot learners," Advances in Neural Information Processing Systems 33, 1877-1901 (2020).

[17] Chen, X., Wang, X., Zhou, J., et al., "Activating more pixels in image super-resolution transformed," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 22367-22377 (2023).

[18] Pourreza, M. and Rafiei, D., "Din-sql: Decomposed in-context learning of text-to-SQL with self-correction," Advances in Neural Information Processing Systems, 36 (2024).

[19] Zhang, B., Ye, Y., Du, G., et al., “Benchmarking the text-to-SQL capability of large language models: A comprehensive evaluation,” arXiv arXiv:2403.02951, (2024).
[20] Li, H., Zhang, J., Li, C., et al., “Resdsql: Decoupling schema linking and skeleton parsing for text-to-SQL,” Proceedings of the AAAI Conference on Artificial Intelligence 37(11), 13067-13075 (2023).
[21] Gu, Z., Fan, J., Tang, N., et al., “Few-shot text-to-sql translation using structure and content prompt learning,” Proceedings of the ACM on Management of Data 1(2), 1-28 (2023).
[22] Liu, A., Hu, X., Wen, L., et al., “A comprehensive evaluation of ChatGPT’s zero-shot Text-to-SQL capability,” arXiv arXiv:2303.13547, (2023).
[23] Li, J., et al., “Can LLM already serve as a database interface? A big bench for large-scale database grounded Text-to-SQLs,” NIPS’23: Proceedings of the 37th International Conference on Neural Information Processing Systems, 42330-4235 (2024).
[24] Yu, T., Zhang, R., Yang, K., et al., “Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task,” arXiv arXiv:1809.08887, (2018).
[25] Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B. and Zhou, J., “Text-to-SQL empowered by large language models: A benchmark evaluation,” arXiv abs/2308.15363, (2023).
[26] Zhao, W., et al., “A survey of large language models,” arXiv arXiv:2303.18223, (2023).
[27] OpenAI, “SQL Translate,” 2024, <https://platform.openai.com/examples/default-sqltranslate> (3 June 2024).
[28] Chang, S. and Fosler-Lussier, E., “How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings,” arXiv arXiv:2305.11853, (2023).
[29] Nan, L., Zhao, Y., Zou, W., et al., “Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies,” arXiv arXiv:2305.12586, (2023).
[30] Taori, R., Gulrajani, I., Zhang, T., et al., “Stanford alpaca: An instruction-following llama model,” https://github.com/tatsu-lab/stanford_alpaca (10-06-2024).
[31] Lewis, P., Perez, E., Piktus, A., et al., “Retrieval-augmented generation for knowledge-intensive nlp tasks,” Advances in Neural Information Processing Systems 33, 9459-9474 (2020).
[32] Vanna, “How Vanna Works,” 2024, <https://vanna.ai/docs/> (3 June 2024).
[33] Zhou, X., Sun, Z. and Li, G., “DB-GPT: Large language model meets database,” Data Science and Engineering, 1-10 (2024).
[34] Trummer, I., “Demonstrating GPT-DB: Generating query-specific and customizable code for SQL processing with GPT-4,” Proceedings of the VLDB Endowment 16(12), 4098-4101 (2023).
[35] Yu, T., Zhang, R., Yang, K., et al., “Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task,” arXiv arXiv:1809.08887, (2018).