Enhancing Text to SQL Generation with Dynamic Vector Search

Indian Statistical Institute, Kolkata
M.Tech. Final Semester,2022-2024
Soumyadweep Mondal

Supervisor : Jayanta Kumar Mukherjee, Technical Architect, ARC Document Solutions
Internal Supervisor : Dipti Prasad Mukherjee, Professor, ISI Kolkata

DECLARATION

I, **Soumyadweep Mondal (Roll No: CrS2217)**, hereby declare that, this report entitled **"Enhancing Text to SQL Generation with Dynamic Vector Search"** submitted to Indian Statistical Institute, Kolkata towards the fulfilment of the requirements for the degree of **Master of Technology** in **CSRU**, is an original work carried out by me under the supervision of **Jyanta Kumar Mukherjee and Dipti Prasad Mukherjee** and has not formed the basis for the award of any degree or diploma, in this or any other institution or university. I have sincerely tried to uphold academic ethics and honesty. Whenever a piece of external information or statement or result is used, then that has been duly acknowledged and cited.

Soumyadweep Mondal

Soumyadweep Mondal

CrS2217
Cryptology & Security
July 12, 2024

Certificate

This is to certify that the work contained in this project report entitled "**Enhancing Text to SQL Generation with Dynamic Vector Search**" submitted by **Soumyadweep Mondal** (Roll No. **CrS2217**) to the Indian Statistical Institute, Kolkata towards the fulfilment of the requirements for the degree of **Master of Technology** in **CSRU** has been carried out by him under my supervision and that it has not been submitted elsewhere for the award of any degree.

[illegible]

Dipti Prasad Mukherjee

Professor,ECSU
Indian Statistical Institute, Kolkata

JAYANTA KUMAR MUKHERJEE

Jayanta Kumar Mukherjee

Technical Architect
ARC Document Solutions India Pvt. Ltd.

ACKNOWLEDGEMENT

I thank everyone who has assisted me in seeing this project through to its completion. I would like first to express my deepest gratitude and deepest regards to my Guides, ARC Document Solutions India Pvt. Ltd. and ISI Kolkata, and sincerely wish to acknowledge their vision, guidance, valuable feedback and constant support throughout this project.
I am indebted to my team members for their steadfast encouragement and time. Lastly, I am grateful to the Indian Statistical Institute, Kolkata, for providing the necessary resources and facilities to complete this project to the best of my ability.

Soumyadweep Mondal

Soumyadweep Mondal

ARC Document Solutions India
Pvt. Ltd.
Cryptology & Security
July 12, 2024

Contents

Abstract

Generating accurate SQL from natural language questions (text-to-SQL) is a long-standing challenge due to the complexities involved in understanding user queries, comprehending database schemas, and generating SQL statements. Traditional text-to-SQL systems have utilized human-engineered solutions and deep neural networks. More recently, pre-trained language models (PLMs) have been employed for text-to-SQL tasks, showing promising results. However, as modern databases and user queries become increasingly complex, the limited comprehension capabilities of PLMs can lead to incorrect SQL generation. This necessitates sophisticated and tailored optimization methods, which restrict the applicability of PLM-based systems.

In contrast, large language models (LLMs) have demonstrated significant advancements in natural language understanding as their scale increases. This thesis explores the integration of LLMs into text-to-SQL systems, highlighting unique opportunities, challenges, and solutions. We propose a novel approach that leverages examples similar to user queries, allowing the model to better understand and generate accurate SQL.

This work provides a comprehensive review of LLM-based text-to-SQL systems, outlining current challenges and the evolutionary process of the field. We introduce datasets and metrics designed for evaluating text-to-SQL systems. Finally, we discuss remaining challenges and propose future directions for research in this domain.

Chapter 1

Introduction

1.1 Background

Text-to-SQL [8],[10] is a long-established task in the research of natural language processing, aiming to translate natural language questions into SQL queries that can be executed on databases. For instance, as illustrated in Figure 1.1 below, a large language model (LLM)-based text-to-SQL system can take a user question like ”What movies are directed by Satyajit Ray?” and the corresponding database schema as input, then generate an SQL query to retrieve the relevant content. In response of the above question the LLM will generate this SQL query ”SELECT Title FROM DIRECTORS_NAME_table WHERE Director = ’Satyajit Ray’”, then based on the retrieved data by executing this SQL query, the user will be given the answer, as shown in the figure below.

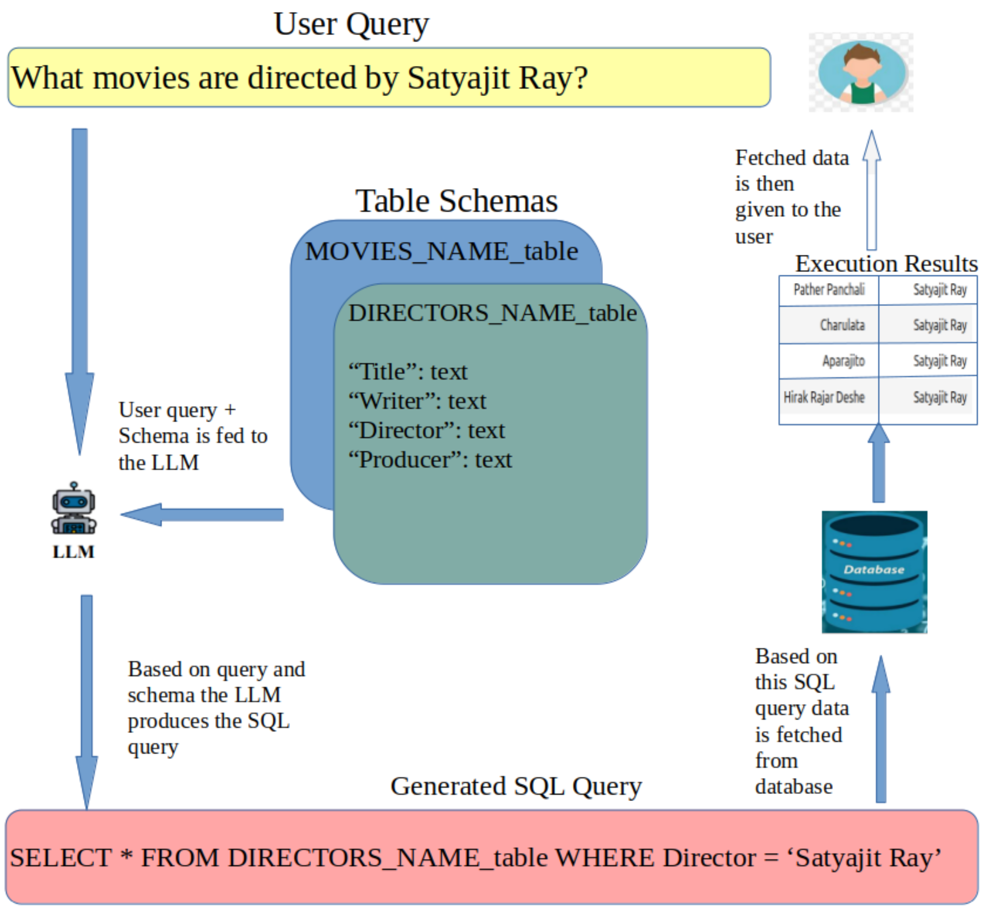

User Query
What movies are directed by Satyajit Ray?
Table Schemas
MOVIES_NAME_table
DIRECTORS_NAME_table
"Title": text
"Writer": text
"Director": text
"Producer": text
User query + Schema is fed to the LLM
LLM
Fetched data is then given to the user
Execution Results
Pather Panchali
Satyajit Ray
Charulata
Satyajit Ray
Aparajito
Satyajit Ray
Hirak Rajar Deshe
Satyajit Ray
Database
Based on query and schema the LLM produces the SQL query
Based on this SQL query data is fetched from database
Generated SQL Query
SELECT * FROM DIRECTORS_NAME_table WHERE Director = 'Satyajit Ray'

Figure 1.1: Diagram of creating SQL query from natural language question using LLM, retrieving data from database using this SQL query, then answering the user with the retrieved data.

Here is the pseudocode and details of how the process happens. Consider the following two tables:

Title	**ID**	**Year**
Pather Panchali	PP55	1955
Charulata	C64	1964
Aparajito	SR56	1956
Meghe Dhaka Tara	MDT60	1960
Hirak Rajar Deshe	HRD80	1980

Table 1.1: Movies Table

DIRECTORS_NAME	ID	Age
Satyajit Ray	PP55	65
Satyajit Ray	C64	65
Satyajit Ray	SR56	65
Ritwik Ghatak	MDT60	65
Satyajit Ray	HRD80	50

Table 1.2: Directors Name Table

Code: Fetch the database schema

```
def get_sqlite_schema(db_path):
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()

    cursor.execute("SELECT name FROM sqlite_master WHERE type='
        ↪ table';")
    tables = cursor.fetchall()

    schema = {}
    for table in tables:
        table_name = table[0]
        cursor.execute(f"PRAGMA table_info({table_name});")
        schema[table_name] = cursor.fetchall()

    conn.close()
    return schema
```

This will give us the following schema information:

```
Table: directors
(0, 'DIRECTORS_NAME', 'VARCHAR(50)', 0, None, 0)
```

```
(1, 'ID', 'VARCHAR(50)', 0, None, 0)
(2, 'Age', 'INTEGER', 0, None, 0)
Table: movies
(0, 'Title', 'VARCHAR(50)', 0, None, 0)
(1, 'ID', 'VARCHAR(50)', 0, None, 0)
(2, 'year', 'INTEGER', 0, None, 0)
```

The following is the pseudocode of the entire process from getting user query to giving the answer to the user.

Code: Fetch the database schema

```
 question = input("Which movies are directed by Satyajit Ray?"
    ↪ )
 schema = get_sqlite_schema(db_path)
 prompt = f"""You are a SQL expert. Given an input question :
    ↪ {question}, create a syntactically correct SQL query to
    ↪ run and return ONLY the generated Query and nothing else
    ↪ . Unless otherwise specified, do not return more than {
    ↪ top_k} rows.
Here is the relevant table info:{schema}"""
sqlquery = LLM.run(prompt)
```

The LLM returns the SQL ”””’**SELECT Title FROM movies**

WHERE ID IN (SELECT ID

FROM directors

WHERE DIRECTORS_NAME = ’Satyajit Ray’)’”””. Then we execute this SQL query to get the following result:

(’Pather Panchali’,), (’Charulata’,), (’Aparajito’,), (’Hirak Rajar Deshe’,)

But if the user asks the question ”Which movies are directed by satyajit ray?” then the generated SQL query is **”””’SELECT Title FROM movies WHERE ID IN (SELECT ID FROM directors WHERE DIRECTORS_NAME = ’satyajit ray’);’””**

When this query is executed it retrieves no data from the database because in the subquery the matching condition is ”WHERE DIRECTORS_NAME = ’satyajit ray’”, but in the Table 1.2, in DIRECTORS_NAME column the directors name is written as ”Satyajit Ray”

That means user have to rightly spell, and they have to mention upper case or lower case words accurately. Which is too much demand from the user, when we know that user may not have accurate information about the database entries. To address the this issue, we will take help of a vector database. We will create possible user query and SQL query pair that covers overall information about all unique database entries in each column. Then whenever a user query comes, we retrieve most similar 10 pairs based on the dense vector similarity measure between the user query and the queries in the vector database. These 10 similar pairs along with the user query is then sent to the LLM. Based on this information the LLM then generates the SQL query. More details regarding this is given in section 3.

1.2 Problem Statement

To generate a SQL query the LLM is given the user query and the schema of the database.
By the discussion in section 1.1, we have the following issues:

(1) Giving away the whole schema of the database means giving a lot of information to the language model. Can we do the same by giving less information?

(2) The SQL query depends on user query. If the user asks ”What movies are directed by Satyajit Ray?”, then the LLM generates SELECT Title FROM movies WHERE ID IN (SELECT ID FROM directors WHERE DIRECTORS NAME = ’Satyajit Ray’), but when user asks ”What movies are directed by satyajit ray?”, then the generated SQL is SELECT

Title FROM movies WHERE ID IN (SELECT ID FROM directors WHERE DIRECTORS NAME = ’satyajit ray’). Here in lies another problem, because in the DIRECTORS_Name column, the name may be saved as ”Satyajit Ray”. In that case when the user asks the ”What movies are directed by satyajit ray?”, the generated SQL query is wrong.

1.3 Research Objectives

We will try to address the following issues:

- To develop methods for generating SQL queries without providing the database schema, thereby minimizing the exposure of sensitive data.
- To address the challenge of generating accurate WHERE clause conditions in SQL queries(e.g. WHERE Director = ”Satyajit Ray” / WHERE Director = ”satyajit ray”), which often require exact string matching, even when the complete schema is available. This research aims to propose solutions to improve the precision and reliability of these conditions.

Chapter 2

Literature Overview

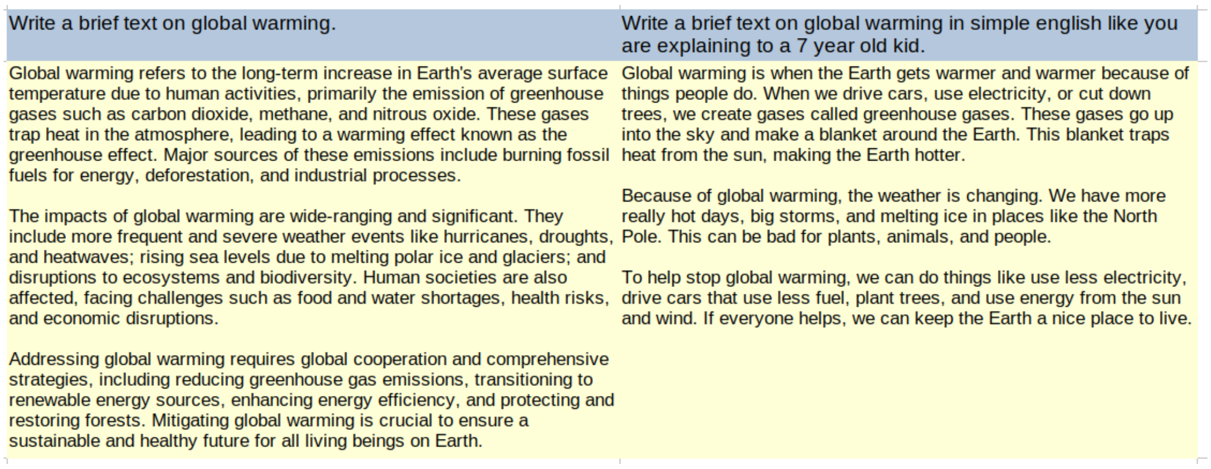
2.1 Context Windows

The extent of text that an LLM can "see" at any given moment is referred to as its context window. In LLMs, text is divided into units called tokens, which can range from single characters to whole words. The size of a model's context window [5] determines the number of tokens it can process at once. Consequently, a larger context window allows the model to consider more tokens, enhancing its understanding of the text and enabling it to generate responses that are more coherent and contextually appropriate. So, basically context window is the number of tokens the LLM can handle at one go. That means **the total number of tokens in prompt + total number of tokens in generated answer = Length of context window.** So, we need to be very careful about prompt designing. If the prompt is too long then the generated answer will be short, hence may be the answer is not good enough. Since, for big database the schema size will be very big. So, sending the schema will cost a lot of tokens, hence the leverage to put our specific instruction in the prompt will be comprised and if the prompt length is more than context window, then LLM will be unable to process it, hence will give error.

2.2 Prompt

A prompt [1] is a piece of text provided to an AI model to guide its response or output. It sets the context and indicates what type of information or action is required from the model.

Prompts are used to initiate the model's text generation process. They help the model understand the task at hand, whether it is answering a question, completing a sentence, translating text, or generating creative content. To understand the effect of prompts [4] on LLM's output see the picture below.



Write a brief text on global warming.	Write a brief text on global warming in simple english like you are explaining to a 7 year old kid.
Global warming refers to the long-term increase in Earth's average surface temperature due to human activities, primarily the emission of greenhouse gases such as carbon dioxide, methane, and nitrous oxide. These gases trap heat in the atmosphere, leading to a warming effect known as the greenhouse effect. Major sources of these emissions include burning fossil fuels for energy, deforestation, and industrial processes. The impacts of global warming are wide-ranging and significant. They include more frequent and severe weather events like hurricanes, droughts, and heatwaves; rising sea levels due to melting polar ice and glaciers; and disruptions to ecosystems and biodiversity. Human societies are also affected, facing challenges such as food and water shortages, health risks, and economic disruptions. Addressing global warming requires global cooperation and comprehensive strategies, including reducing greenhouse gas emissions, transitioning to renewable energy sources, enhancing energy efficiency, and protecting and restoring forests. Mitigating global warming is crucial to ensure a sustainable and healthy future for all living beings on Earth.	Global warming is when the Earth gets warmer and warmer because of things people do. When we drive cars, use electricity, or cut down trees, we create gases called greenhouse gases. These gases go up into the sky and make a blanket around the Earth. This blanket traps heat from the sun, making the Earth hotter. Because of global warming, the weather is changing. We have more really hot days, big storms, and melting ice in places like the North Pole. This can be bad for plants, animals, and people. To help stop global warming, we can do things like use less electricity, drive cars that use less fuel, plant trees, and use energy from the sun and wind. If everyone helps, we can keep the Earth a nice place to live.

Figure 2.1: An illustration showing general prompt on the left and a specific and precise prompt on the right, generated by GPT-4o

2.3 Prompt Principles and Guides

Bsharat et.al. [6] tabulate 26 ordered prompt principles, which can further be categorized into five distinct categories, as shown in the table below.

- **Prompt Structure and Clarity:** Integrate the intended audience in the prompt.
- **Specificity and Information:** Implement example-driven prompting (Use few-shot prompting)
- **User Interaction and Engagement:** Allow the model to ask precise details and requirements until it has enough information to provide the needed response.

- **Content and Language Style:** Instruct the tone and style of response.
- **Complex Tasks and Coding Prompts:** Break down complex tasks into a sequence of simpler steps as prompts.

Category	Principles
Prompt Structure and Clarity	Integrate the intended audience in the prompt. Employ affirmative directives such as ‘do’ while steering clear of negative language like ‘don’t’. Use Leading words like writing “think step by step.” Use output primers, which involve concluding your prompt with the beginning of the desired output. Use Delimiters. When formatting your prompt, start with ‘###Instruction###’, followed by either ‘###Example###’ or ‘###Question###’ if relevant. Subsequently, present your content. Use one or more line breaks to separate instructions, examples, questions, context, and input data.
Specificity and Information	Implement example-driven prompting (Use few-shot prompting). When you need clarity or a deeper understanding of a topic, idea, or any piece of information, utilize the following prompts: o Explain [insert specific topic] in simple terms. o Explain to me like I’m 11 years old. o Explain to me as if I’m a beginner in [field]. o “Write the [essay/text/paragraph] using simple English like you’re explaining something to a 5-year-old.” Add to your prompt the following phrase “Ensure that your answer is unbiased and avoids relying on stereotypes.” To write any text intended to be similar to a provided sample, include specific instructions: o “Use the same language based on the provided paragraph [/title/text/essay/answer].” When you want to initiate or continue a text using specific words, phrases, or sentences, utilize the provided prompt structure: o I’m providing you with the beginning [song lyrics/story/paragraph/essay...]: [Insert lyrics/words/sentence]. Finish it based on the words provided. Keep the flow consistent. Clearly state the model’s requirements that the model must follow in order to produce content, in form of the keywords, regulations, hint, or instructions. To inquire about a specific topic or idea and test your understanding g, you can use the following phrase : o “Teach me the [Any theorem/topic/rule name] and include a test at the end, and let me know if my answers are correct after I respond, without providing the answers beforehand.” To write an essay/text/paragraph/article or any type of text that should be detailed: o “Write a detailed [essay/text/paragraph] for me on [topic] in detail by adding all the information necessary.

Table 2.1: Prompt Principle Categories [6]

Category	Principles
User Interaction and Engagement	Allow the model to elicit precise details and requirements from you by asking you questions until he has enough information to provide the needed output o “From now on, I would like you to ask me questions to ...” To write an essay /text /paragraph /article or any type of text that should be detailed: “Write a detailed [essay/text/- paragraph] for me on [topic] in detail by adding all the necessary information.”
Content and Language Style	To correct/change specific text without changing its style: “Try to revise every paragraph sent by users. You should only improve the user’s grammar and vocabulary and make sure it sounds natural. You should maintain the original writing style, ensuring that a formal paragraph remains formal.” Incorporate the following phrases: “Your task is” and “You MUST.” Incorporate the following phrases: “You will be penalized.” Assign a role to the language model. Use the phrase “Answer a question given in natural language form” in your prompts. No need to be polite with LLM so there is no need to add phrases like “please”, “if you don’t mind”, “thank you”, “I would like to”, etc., and get straight to the point. Repeat a specific word or phrase multiple times within a prompt. Add “I’m going to tip $xxx for a better solution!”
Complex Tasks and Coding Prompts	Break down complex tasks into a sequence of simpler prompts in an interactive conversation. When you have a complex coding prompt that may be in different files: o “From now and on whenever you generate code that spans more than one file, generate a [programming language] script that can be run to automatically create the specified files or make changes to existing files to insert the generated code. [your question].” Combine Chain-of-thought (Cot) with few-shot prompts.

Table 2.2: Prompt Principle Categories [6]

2.3.1 Zero-Shot Experiments

In the realm of text-to-SQL parsing, in-context learning has demonstrated promising results. Current in-context learning approaches, such as those by Tai et all [7], frequently utilize techniques like chain-of-thought (COT) prompting [6],[2] to tackle complex text-to-SQL tasks. Our experiments employing single-step COT with GPT-3.5 reveal that, while COT can somewhat broaden the problem-solving capabilities of large language models (LLMs), the effectiveness of these single-step prompts remains constrained. Specifically, instructions embedded within lengthy prompts are often overlooked due to the LLM's attention being spread thinly across the extensive text. Furthermore, enhancing prompt text typically leads to a seesaw effect, complicating the improvement of overall performance, as enhancements are not consistently effective across different types of problems.

Here is an high level overview of how a Chain of Thought prompt [2],[6],[7] looks like:

- **Understand the Goal**

  (1) The goal is to retrieve specific information ([desired columns]) from the [table name] table

  (2)We need to apply a condition ([condition]).

- **Identify the Relevant Table and Columns**

  (1) Table: [table name].

  (2) Columns: [desired columns].

- **Construct the Base Query**

  (1)Start with selecting the necessary columns:SELECT [desired columns] FROM [table name]

- **Add the Condition**

  (1)We need to filter rows based on the condition ([condition])

(2)Use the appropriate SQL syntax to specify the condition.

- **Complete the SQL Query**

  (1) Combine all parts to form the final query: SELECT [desired columns] FROM [table name] WHERE [condition];

2.3.2 Limitations of Zero-Shot Learning

One significant drawback of zero-shot learning in SQL query generation is its limited efficacy with complex queries. The core issue lies in field matching accuracy. For instance, consider the query: ”What movies are directed by Satyajit Ray?”. The correct SQL query could either be: ”SELECT Title FROM DIRECTORS_TABLE WHERE Director = ’Satyajit Ray’” or ”SELECT Title FROM DIRECTORS_TABLE WHERE Director = ’satyajit ray’”.
The model lacks knowledge about the specific entries within the database columns, making it highly dependent on the user’s input query. For example, if the user inputs, ”What movies are directed by Satyajit Ray?”, the generated query will be:
”SELECT Title FROM DIRECTORS_TABLE WHERE Director = ’Satyajit Ray’”.
Conversely, if the user inputs, ”What movies are directed by satyajit ray?”, the generated query will be:
”SELECT Title FROM DIRECTORS_TABLE WHERE Director = ’satyajit ray’”.
This dependence on exact user input highlights a significant limitation: the user must possess knowledge of the precise database entries. This requirement undermines the user-friendliness and practicality of zero-shot learning for SQL query generation, especially for users unfamiliar with the specific data formatting within the database.

2.3.3 Few-Shot Prompt

Few-shot prompting [1] involves providing a model with a prompt (in-context learning) along with a few task-related examples to guide the model towards better performance. This technique allows the model to learn quickly and generalize across various tasks.

A major advantage of few-shot prompting is its efficiency in using limited annotated data, as opposed to requiring large datasets for training. This method supports rapid prototyping and deployment of NLP systems, making it highly suitable for industry applications. Additionally, it enables the model to continually learn and adapt with minimal supervision, further enhancing its capabilities.

2.3.4 Few Shot Improvement

In the context of text-to-SQL tasks, few-shot prompting improved over zero-shot prompting due to several key factors:

1. Contextual Understanding:

- Zero-shot prompting requires the model to generate SQL queries without any specific examples related to the task. The model relies solely on its pre-trained knowledge and the provided prompt, which can be challenging for complex or ambiguous queries.
- Few-shot prompting, on the other hand, provides the model with a few examples of input-output pairs. These examples help the model understand the specific format and structure expected for the task, enhancing its ability to generate accurate SQL queries.

2. Pattern Recognition:

- With few-shot examples, the model can recognize patterns in the examples provided. This helps it understand how to map natural language questions to SQL queries more effectively.
- Zero-shot scenarios, the model has to infer the required patterns and mappings from scratch, which can lead to more errors and less accurate query generation.

3. Clarification of Ambiguities:

- Text-to-SQL tasks often involve understanding the schema of the database and the relationships between tables. Few-shot prompting provides explicit examples of how to handle specific database schemas, reducing ambiguity.
- Without examples, zero-shot models may struggle to correctly interpret and generate queries for unfamiliar or complex schemas.

4. Learning from Similar Cases:

- Few-shot prompting leverages similar cases to illustrate the correct approach to the task. This helps the model generalize better to new but similar queries by drawing parallels from the provided examples.
- zero-shot prompting, the model has no such references, making it harder to generalize accurately from its pre-existing knowledge.

2.3.5 Limitations of Few Shot

While few-shot learning enhances performance in text-to-SQL tasks, several challenges remain. The problem of exact string matching in the WHERE clause has seen improvement; the model, informed by few-shot examples, can recognize specific entries. However, two significant issues persist.

Firstly, the model continues to generate incorrect queries for complex and vague user questions. This issue arises because intricate queries and the diverse nature of user inquiries demand extensive knowledge about the types of entries in each column. Without a comprehensive understanding of the database entries, the model struggles to accurately interpret and construct the necessary SQL queries.

Secondly, determining the optimal number of examples to include in the prompt poses a challenge. The context window of large language models (LLMs) limits the number of examples that can be effectively used. This limitation restricts the model's ability to fully grasp the diversity of potential queries, thereby impacting its performance.

Addressing these issues is crucial for further advancing the effectiveness of few-shot learning in text-to-SQL applications.

Chapter 3

Method

To address the aforementioned issues, we adopted a novel approach that diverges from the traditional methods discussed earlier and in the literature. Instead of directly providing the database schema to the model for SQL generation, we implemented a dynamic few-shot learning technique enhanced with vector search. The steps of our approach are as follows:

(i) **Generating Query-Response Pairs:** We created possible pairs of user queries (τ) and corresponding SQL queries (σ). These pairs were generated in a manner that ensured the coverage of unique column entries.

(ii) **Populating the Vector Database:** The generated (τ, σ) pairs were stored in a vector database.

(iii) **Retrieving Similar Pairs:** When a user submits a query, the system retrieves the top 10 most similar (τ, σ) pairs from the vector database based on the query.

(iv) **Informing SQL Generation:** These retrieved pairs provide the model with sufficient information regarding the formation of the WHERE clause in the SQL query.

By leveraging this dynamic few-shot learning approach combined with vector search, the model gains a better understanding of the database entries and the diversity of potential queries, leading to more accurate SQL generation.

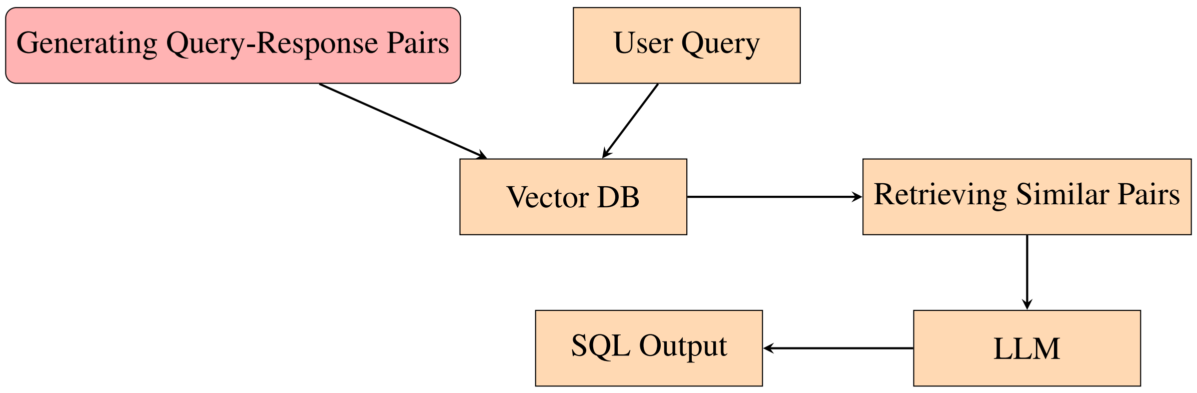


Figure 3.1: Workflow Diagram

Now we will describe the above steps in detail. To understand the process of (τ, σ) pair generation, let us consider the following example.

Building	**Course**	**Teacher**	**Floor**	**Degree**
SN Bose	ML	A Ghosh	9th floor	M Tech 1
SN Bose	ML	B Ghosh	9th floor	M Tech 2
SN Bose	DL	A Ghosh	9th floor	M Tech 1
SN Bose	DL	B Ghosh	9th floor	M Tech 2

Table 3.1: Example Database

We can clearly observe that tha table has a lot of redundant information. We don't want to flood the vector database with redundant information, because if there are many redundant information then the similarity search of step (iii) will retrieve very little unique data, hence the model will only get very little new information (knowledge about database), which is a loss of tokens. So we carefully design some questions and their SQL query so that the query covers almost all the unique column elements of each coloumn of the table.

Q: Classes taken by Teacher_Name

Pseudocode

```
Unique_Teachers = [Unique Entries of The Column 'Teacher']

qa_pair = []
```

```
for teacher in Unique_Teachers:
    Question = f"Classes taken by {teacher}"
    SQL = f"SELECT Course, Degree, Building, Floor FROM Example
      ↪  Database WHERE Teacher = '{teacher}'"
    qa_pair.append(zip(Question, SQL))
```

In the pseudo code above only unique entries of the column ”Teacher” gets included. Only the following two pairs will be included in the database:

1. {'Question':'Classes taken by A Ghosh', 'SQL':'SELECT Course, Degree, Building, Floor FROM Example Database WHERE Teacher = 'A Ghosh' ' }

2.{'Question':'Classes taken by A Ghosh', 'SQL':'SELECT Course, Degree, Building, Floor FROM Example Database WHERE Teacher = 'B Ghosh' ' }

Now suppose the user asks a question like: ” Which course is taught by a ghosh?”. Look that this question is not same as any of the above two questions that we pushed into the vector database. Another notable thing is that in the question ” Which course is taught by a ghosh?” the teacher's name is written as 'a ghosh' not 'A Ghosh'. But since we perform a dense vector vector search of user query ” Which course is taught by a ghosh?” with the questions in the vector database. It will retrieve the pair :

{'Question':'Classes taken by A Ghosh', 'SQL':'SELECT Course, Degree, Building, Floor FROM Example Database WHERE Teacher = 'A Ghosh' ' }

From this example the model will be able to understand that in the 'Teacher' column the name of the teacher is included as 'A Ghosh' and the columns where course details is given are named as 'Course', 'Degree'. Hence the model will be able to generate the following SQL query: ”SELECT Course, Degree FROM Example Database WHERE Teacher='A Ghosh' ”.

3.1 Results

We have evaluated the text to SQL generation result on WikiSQL dataset. This is the WikiSQL Leaderboard:

Model	**Test Logical Form Accuracy**
SeaD +Execution-Guided Decoding (Xu 2021) (Ant Group, Ada & ZhiXiaoBao)	87.5
SDSQL +Execution-Guided Decoding (Hui 2020) (Alibaba Group)	87.0
IE-SQL +Execution-Guided Decoding (Ma 2020) (Ping An Life, AI Team)	87.8

Table 3.2: Test Logical Form Accuracy for Various Models

The top two models [9],[3] of the leader board both uses schema aware learning. With schema aware learning approach they achieve those results. But our approach is different due to the fact that since we are using pretrained models, that may not be an exact fit for our specific use case. So, we cannot totally rely on the model also, due to the fact that we have no control over users query. So, adding question, SQL pair to database gives us some control over the failed queries. Because, monitoring the failed queries we can add some more examples on the database to handle that type of queries.

WikiSQL dataset has train, validation and test split. Train split contains 56355 question, SQL pair. Which we have pushed into database. Validation split contains 8421 question, SQL pair. Since in our case we don't train a model and no need of hyperparameter tuning we omit this split. Test split contains 15878 question, SQL pair. For each question in test split, we retrieve 10 most related sample question sql pair of the train split, and based on that the model generates a SQL query.

We want to evaluate the accuracy of SQL generation. Now the same SQL query can be written in different ways. So, it is very time consuming for human evaluation of SQL generation for a dataset of length 15878. So, we leverage the code understanding and reasoning capability of GPT-4 to measure the quality of generated SQL against the given sql in the dataset and assign a score between 0 and 1.This gives us the following result.

We are able to reach test logical form accuracy of 85.56%.

Chapter 4

Conclusion & Future Works

4.1 Conclusion

Although we have not yet surpassed the top results on the leaderboard, our performance is very close. More importantly, we have successfully addressed a significant issue that previously affected our results. We now have the capability to monitor which types of user queries are failing. Based on this analysis, we can add specific (τ, σ) pairs to our database, thereby improving our model's accuracy. This capability is particularly vital for enterprises like ours that prioritize serving their users effectively.

4.2 Future Works

Our approach primarily aims to address two issues: (1) minimizing the exposure of sensitive data, such as schema information, and (2) generating accurate WHERE clause conditions in SQL queries. We have made significant progress in addressing the second issue. Although we do not provide the complete database schema, we include some column entries for the model. However, without the full schema, these column entries may lack context, which means data leakage is not completely mitigated. To fully address this problem, we propose using masking or cryptographic techniques to protect sensitive information while still enabling the model to generate accurate SQL queries.

Bibliography

[1] Prompt engineering. *https://en.wikipedia.org/wiki/Prompt*$_e ngineering.$

[2] Rishi Bommasani Colin Raffel Barret Zoph Sebastian Borgeaud Dani Yogatama Maarten Bosma Denny Zhou Donald Metzler et al. ason Wei, Yi Tay. Emergent abilities of large language models. *https://openreview.net/pdf?id=yzkSU5zdwD*, 2022.

[3] Ruiying Geng Binhua Li Yongbin Li † Jian Sun Alibaba Group Binyuan Hui , Xiang Shi. Improving text-to-sql with schema dependency learning. *https://arxiv.org/pdf/2103.04399*, Dec 10, 2021.

[4] Jules S. Damji. Best prompt techniques for best llm responses. *The Modern Scientist, https://medium.com/the-modern-scientist/best-prompt-techniques-for-best-llm-responses-24d2ff4f6bca*, Feb 12, 2024.

[5] Chandler K. Context windows: The short-term memory of large language models. *Medium, https://medium.com/@crskilpatrick807/context-windows-the-short-term-memory-of-large-language-models-ab878fc6f9b5*, Nov 4, 2023.

[6] Zhiqiang Shen Sondos Mahmoud Bsharat, Aidar Myrzakhan. Principled instructions are all you need for questioning llama-1/2, gpt-3.5/4. *https://arxiv.org/pdf/2312.16171*, 18 Jan 2024.

[7] Chang-You Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. Exploring chain-of-thought style prompting for text-to-sql. 2023. https://arxiv.org/abs/2305.14215.

[8] Richard Socher Victor Zhong, Caiming Xiong. Generating structured queries from natural language using reinforcement learning. *https://arxiv.org/pdf/1709.00103*, 2017.

[9] Kuan Xu, Yongbo Wang, Yongliang Wang, Zujie Wen, and China Yang Dong Ant Group Hangzhou. Sead: End-to-end text-to-sql generation with schema-aware denoising. *https://arxiv.org/pdf/2105.07911*, Jan 30, 2023.

[10] Qinggang Zhang2 Hao Chen2 Junnan Dong2 Feiran Huang1 Zijin Hong1, Zheng Yuan2 and Xiao Huang2. Next-generation database interfaces: A survey of llm-based text-to-sql. *https://arxiv.org/pdf/2406.08426*, 2024.