Team_Optimist at EHRSQL 2024: Text-to-SQL Generation using Large Language Model for EHR Analysis

Sourav Bhowmik Joy Rohan Redwan Argha Pratim Saha
Minhaj Ahmed Utsho Das Partha Sarothi Bhowmik
CSE, Shahjalal University of Science & Technology

Abstract

This paper explores the application of the sqlcoders model, a pre-trained neural network, for automatic SQL query generation from natural language questions. We focus on the model's internal functionality and demonstrate its effectiveness on a domain-specific validation dataset provided by EHRSQL. The sqlcoders model, based on transformers with attention mechanisms, has been trained on paired examples of natural language questions and corresponding SQL queries. It takes advantage of a carefully crafted prompt that incorporates the database schema alongside the question to guide the model towards the desired output format.

1 Introduction

Electronic health records (EHRs), a large collection of data related to digital medical records, serve as the backbone of modern healthcare, storing a wealth of patient information. This data, encompassing diagnoses, procedures, medications, and more, offers invaluable insights for clinical decision-making and research.[3] [4] However, effectively utilizing this vast resource is often hampered by the complexity of querying the underlying relational databases.

Traditionally, hospital staff relies on pre-defined rule conversion systems to interact with EHR databases. These systems, while functional, limit access to information beyond pre-configured rules. Modifying and extending these systems requires specialized training, creating a bottleneck for users seeking broader data access.

This paper explores the potential of natural language processing (NLP) to bridge this gap. We present a system that leverages the power of large language models (LLMs) to automatically translate natural language questions into corresponding SQL queries. This approach empowers users to directly query the EHR database using natural language, eliminating the need for complex SQL syntax and significantly streamlining data retrieval.

The core of our system lies in a pre-trained LLM, specifically the sqlcoders model. This model, trained on paired examples of natural language questions and their corresponding SQL queries, has the remarkable ability to understand the user's intent and translate it into the appropriate database query language. We delve into the inner workings of the sqlcoders model and the concept of prompt engineering, a crucial aspect of guiding the LLM towards generating accurate SQL statements.

By focusing on open-source LLMs like sqlcoders, our work contributes to the broader exploration of readily available resources for NLP tasks. We aim to demonstrate the effectiveness of supervised fine-tuning in enhancing the performance of open-source LLMs for the challenging task of text-to-SQL translation in the specific domain of healthcare.

This paper is structured as follows. First, we discuss related work on text-to-SQL translation, highlighting the advantages of LLM-based approaches and the importance of prompt engineering. Subsequently, we introduce the sqlcoders model and

its methodology. We then present our approach and implementation details, followed by an evaluation of the system's performance. Finally, we conclude by discussing the implications of our work and outlining future directions.

2 Related Work

Extracting SQL queries from natural language questions has been a well-studied area within NLP, with applications spanning various domains. This section explores relevant research directions and highlights how the sqlcoders model aligns with these methodologies.

2.1 Semantic Parsing and Question Answering

Question-to-SQL generation can be viewed as a sub-task of semantic parsing, where the goal is to translate natural language into a formal representation like SQL. Early approaches relied on rule-based systems or semantic parsing methods that focused on identifying the SQL structure and filling slots with relevant information from the question .These methods achieve good performance but struggle with complex queries or domain-specific terminology. [2][6] [7][8]

2.2 Sequence-to-Sequence Learning

Another approach leverages sequence-to-sequence (Seq2Seq) models with attention mechanisms.[5] These models encode the natural language question and decode the corresponding SQL query directly. While effective, they may struggle with order-sensitive aspects of SQL syntax and require large amounts of training data.

2.3 Template-Based Methods and Prompt Engineering

Some studies adopt template-based approaches where pre-defined SQL templates are filled with question elements. While this method can handle complex queries efficiently, it relies heavily on hand-crafted templates and may not generalize well to unseen scenarios. Recent work focuses on "prompt engineering," which involves carefully crafting prompts that guide large language models (LLMs) towards generating the desired output format. The sqlcoders model aligns with this approach by utilizing a comprehensive prompt that incorporates the database schema alongside the question to improve its SQL generation capabilities.

The sqlcoders model addresses the limitations of traditional semantic parsing and Seq2Seq methods by leveraging the power of LLMs. Its ability to learn from paired examples of natural language questions and their corresponding SQL queries allows it to capture complex relationships and generate accurate SQL statements. Additionally, the focus on prompt engineering ensures that the model effectively utilizes the provided database schema information. Compared to template-based methods, the sqlcoders model is more flexible and can potentially adapt to unseen scenarios. However, similar to other LLM-based approaches, it requires careful fine-tuning for optimal performance in the specific domain of healthcare.

3 Methodology

This section delves into the research methodology employed to investigate the effectiveness of the sqlcoders model for automated SQL query generation from natural language questions in the healthcare domain. We exploit the model's capability to learn intricate relationships between natural language and database structures, coupled with the power of prompt engineering, to achieve this goal.

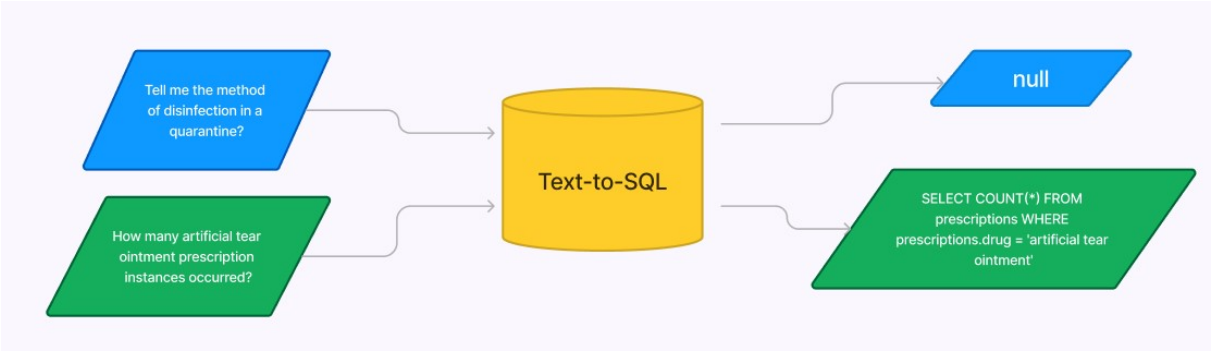

Figure 1: Natual language question to appropriate sql

3.1 Data Preparation

3.1.1 EHR Dataset:

We utilize a well-structured Electronic Health Records (EHR) dataset, namely MIMIC-IV dataset, [1] containing various tables (e.g., patients, medications, diagnoses) and attributes (e.g., patient ID, diagnosis code, medication name) relevant to patient information. This dataset serves as the underlying data source for generating and evaluating SQL queries.

3.1.2 Question-SQL Pairs:

We create a collection of question-SQL pairs specific to the healthcare domain. Each pair consists of a natural language question seeking information from the EHR data and its corresponding valid SQL query that retrieves the desired answer. Here, we can introduce an image (Figure 1) to visually represent a sample question-SQL pair.

3.2 The sqlcoders Model

The core component of our system is the sqlcoders model, a pre-trained large language model (LLM) specifically designed for text-to-SQL translation tasks. Here, we can delve into the mathematical intuition behind the model's functionality, but due to the potentially complex nature of LLM architectures, a high-level explanation might be more suitable for this section.

3.3 Conceptual Framework

The sqlcoders model can be thought of as a function that maps a natural language question (q) and a database schema description (s) to a corresponding SQL query (y). We can represent this mathematically as:

$$y = f(q, s) \tag{3.1}$$

where f represents the models functionality. This function involves a complex neural network architecture, namely transformers with attention mechanisms. During training, the model is exposed to numerous paired examples of questions, schema descriptions, and their corresponding SQL queries. This training process allows the model to develop an internal representation that captures the intricate relationships between:

- **Natural Language Semantics:** The model identifies and encodes the meaning of words and phrases within the natural language question. This includes understanding the intent of the question (e.g., retrieval, aggregation), the entities of interest (e.g., patients, medications), and the relationships between them.
- **Database Schema Knowledge:** The model learns to represent and utilize the information provided in the schema description. This includes understanding the structure of the

database (tables, attributes, data types), the relationships between tables (foreign keys), and the available data elements relevant to answering the question.

- **SQL Constructs and Syntax:** Eventually The model attempts to map the extracted meaning from the question and schema to the appropriate SQL constructs. This includes generating the core components of a query like SELECT, FROM, WHERE, and JOIN, as well as populating them with relevant attributes and conditions based on the question and schema information.

3.4 Prompt Engineering

A crucial aspect of using the sqlcoders model effectively is prompt engineering. We design a comprehensive prompt that incorporates the following elements:

- **Task Description:** This clarifies the task as generating an SQL query to answer the provided question.
- **Question Placeholder:** This section is denoted by a placeholder (e.g., [QUESTION]question[/QUESTION]) where the actual natural language question is inserted during query generation.
- **Schema Description:** This section provides a representation of the database schema, including table names, attributes, and data types. This information is essential for the model to understand the available data and construct valid SQL queries. We can consider different ways to represent the schema, such as tables with columns or a more natural language-like description.
- **Instructions:** We can optionally include instructions for the model, such as handling situations where data might be unavailable or specifying calculations for revenue or cost. These instructions further guide the model towards generating accurate and relevant SQL queries.
- **Answer Placeholder:** This section (e.g., [SQL]) serves as a placeholder where the model will generate the predicted SQL query. In case of an unanswerable question, the model would generate "null" as the answer.

By effectively combining these elements within the prompt, we provide context and guide the sqlcoders model towards generating accurate and relevant SQL statements that retrieve the intended information from the EHR data.

3.5 Query Generation Process

1. **Iterating Through Questions:** We iterate through the collection of natural language questions in the prepared dataset.
2. **Prompt Construction:** For each question, a prompt is constructed by inserting the question into the designated placeholder within the pre-defined prompt template. The constructed prompt and schema description are fed to the sqlcoders model.
3. **Model Prediction:** The model utilizes its learned knowledge and the provided context to predict the most likely sequence of tokens representing a valid SQL query that answers the question.

4 Results

This section dives deeper into the model's performance based on the Reward Scoring (RS) schemes employed for evaluation.

4.1 Evaluation Criteria

The model's effectiveness was assessed using four RS (Reliability Score) schemes,

each representing a different level of penalty for incorrect predictions:

$$\phi_c(x) = \begin{cases} 1 & \text{if } x \in \mathcal{Q}_{ans};\ g(x) = 1;\ Acc(x) = 1 \\ 0 & \text{if } x \in \mathcal{Q}_{ans};\ g(x) = 0, \\ -c & \text{if } x \in \mathcal{Q}_{ans};\ g(x) = 1;\ Acc(x) = 0 \\ -c & \text{if } x \in \mathcal{Q}_{una};\ g(x) = 1, \\ 1 & \text{if } x \in \mathcal{Q}_{una};\ g(x) = 0. \end{cases}$$

Figure 2: The formal definition of RS for a single data instance

- **RS(0):** This is the most lenient scenario where the model receives no penalty for mistakes (c=0). In the context of question answering (QAs) alone, this score essentially reflects execution accuracy in the standard text-to-SQL task.
- **RS(5):** This scenario introduces a moderate penalty (c=5). A correct prediction earns a +1 reward, while each mistake incurs a -5 penalty. In simpler terms, every five accurate predictions compensate for one incorrect prediction.
- **RS(10):** This is considered the primary evaluation metric (c=10). Each correct prediction earns a +1, whereas each mistake results in a -10 penalty. This means ten correct predictions are needed to outweigh a single incorrect prediction.
- **RS(N):** This scenario represents the most stringent evaluation (c=N, where N is the size of the evaluation data). Here, even a single mistake can lead to a negative overall score, even if all other predictions (N-1) are correct.

4.2 Model Performance

The model's performance varied significantly across the different RS schemes:

1. **RS(0):** **14.14** - This positive score in the most lenient scenario indicates that the model can generate some correct SQL queries. However, the lack of penalty for mistakes doesn't provide a clear picture of its true accuracy.
2. **RS(5):** **-349.61** - The substantial drop in score compared to RS(0) suggests a high number of incorrect predictions. The moderate penalty magnifies these errors, highlighting the model's sensitivity to mistakes.
3. **RS(10):** **-713.37** - This significantly lower score further emphasizes the model's shortcomings. With a stricter penalty, the negative impact of errors becomes even more pronounced.
4. **RS(N):** **-84885.86** - The negative score under the most stringent evaluation highlights severe limitations. Even if the model generates a large number of correct queries, a single mistake can significantly impact the overall performance.

RS Scheme	Score
RS(0) (No Penalty)	**14.14**
RS(5) (Moderate Penalty)	**-349.61**
RS(10) (Main Evaluation Metric)	**-713.37**
RS(N) (Strict Penalty)	**-84885.86**

Table 1: Model Performance under Different Reliability Scoring (RS) Schemes

4.3 Key Findings

The model's inability to achieve positive scores under most RS scenarios indicates a fundamental limitation in generating accurate SQL queries.

The significant drop in score with increasing penalty severity demonstrates the model's susceptibility to errors. Even a moderate level of penalty leads to substantial performance degradation.

The stark contrast between RS(0) and other scores emphasizes the importance of incorporating penalties into model evaluation. It provides a more realistic assessment of the model's ability to handle real-world scenarios with potential errors. Moreover, the negative RS(N) score reveals a lack of robustness. Even a single mistake can outweigh a large number of correct predictions, indicating the model's inability to consistently generate reliable queries.

5 Conclusion and Future Direction

This investigation evaluated the sqlcoder model's performance in generating SQL queries using various Reliaibilty Scoring (RS) schemes. Though the model shows a basic capability to generate some correct results (evident in the positive RS(0) score), its overall accuracy and robustness require significant improvement. .By focusing on exploration of different model architectures, enhanced error handling and incorporating human expertise, future investigations hold promise for significant advancements in this domain.

Acknowledgments

We extend our sincere gratitude to the contributors of the SQLCoder repository (https://github.com/defog-ai/sqlcoder). We acknowledge the authors contributions in this field without which this paper would not have been possible.

References

[1] Alistair E. W. Johnson, Lucas Bulgarelli, Lu Shen, Alvin Gayles, Ayad Shammout, Steven Horng, Tom J. Pollard, Sicheng Hao, Benjamin Moody, Brian Gow, Li-wei H. Lehman, Leo A. Celi, and Roger G. Mark. Mimic-iv, a freely accessible electronic health record dataset. *Scientific Data*, 10(1):1, 2023.

[2] Dongjun Lee, Jaesik Yoon, Jongyun Song, Sanggil Lee, and Sungroh Yoon. One-shot learning for text-to-sql generation, 2019.

[3] Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. Ehrsql: A practical text-to-sql benchmark for electronic health records. 35:15589–15601, 2022.

[4] Gyubok Lee, Sunjun Kweon, Seongsu Bae, and Edward Choi. Overview of the ehrsql 2024 shared task on reliable text-to-sql modeling on electronic health records. In *Proceedings of the 6th Clinical Natural Language Processing Workshop*, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

[5] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[6] Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning, 2017.

[7] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. Typesql: Knowledge-based type-aware neural text-to-sql generation, 2018.

[8] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning, 2017.