Leveraging LLMs for SQL Generation in Computational Biology

Raj Janardhan, Colin Naughton, Jeff Kramer, Saikanam Joaddar Siam

1 Abstract

Advancements in Large Language Models (LLMs) have revolutionized automatic code generation and provide opportunities for database management. This project explores the use of LLM agents for generating SQL queries, presenting an intuitive method for naive users to interact with complex databases. We fine-tune models, such as Llama-2-7B and Mistral-7B, using Gretel AI's text-to-SQL dataset and employ chain-of-thought prompt engineering to produce a SQL agent that generates accurate, concise SQL queries spanning multiple tables of a fake company's MySQL database[3-5]. We evaluate our agent using metrics including the query compilation accuracy, query output accuracy, and query verbosity. Results show that the combination of prompt engineering and fine-tuning produces SQL agents with superior evaluation metrics than agents produced using either method alone. Our lightweight, fine-tuned SQL agent can translate complex user questions into concise, accurate queries and serves as a useful database tool for non-technical users. In terms of applications in computational biology, medical records and databases, such as the AoU database, hold critical information that can be hard to decipher. Creating domain-specific agents for SQL generation can help in the space of biology due to its ability to ease data collection and aggregation.

2 Introduction

Interactions between users and databases are frequently hindered by lack of technical understanding of Structured Query Language, or SQL[7]. The complexity of SQL results in a steep learning curve to achieve even moderate levels of understanding and serves as a gap between databases and users. Advancements in LLMs have established an agent-based framework for bridging this gap through prompt engineering and models fine-tuned on large-scale SQL datasets.

This project investigates the use of LLMs as SQL agents to facilitate user-database interactions. By using lightweight models such as Llama-2-7B and Mistral-7B finetuned on a specialized SQL data set, we aimed to create a tool to allow non-technical users to easily interact with complex databases. Our approach leverages Chain-of-Thought prompt engineering coupled with fine tuning techniques to optimize our agent's ability to produce accurate, concise SQL queries given a user's question. Our resulting SQL agent provides an intuitive way for users to interact with complex databases without requiring any knowledge of SQL.

3 Related Work

The use of LLMs to interact with databases is heavily researched. Recent work has shown the benefit of combining pre-trained language models focused on SQL query generation with LLMs such as chatGPT to produce more accurate queries than either alone (1). Other work has made progress in this area by creating graph representations of databases and queries, creating linkages between the query and data schema nodes, and passing info about these linkages as part of a SQL-generating

LLM prompt (2).

Recent advancements in LLMs have significantly impacted code generation and debugging, as it is demonstrated by the study in [4]. The study introduced the concept of Self-Debugging, where LLMs debug their own code without human feedback using a method known as rubber duck debugging. This approach has shown good improvements in code generation benchmarks including text-to-SQL tasks on the Spider dataset.

The integration of LLMs with database systems has been further explored to improve user interactions with databases. The paper titled "DB-GPT: Empowering Database Interactions with Private Large Language Models" [5] presents a system that utilizes LLMs to interpret natural language queries and generate SQL queries. The methodology here leverages a novel retrieval augmented generation knowledge system paired with an adaptive learning mechanism. This work improves the intuitiveness and security of database interactions.

Various approaches have been proposed to bridge the gap between users and databases. The work presented in [6] talks about a multitask pretraining framework that enhances the performance of LLMs in generating SQL by incorporating context information from database schemas. Furthermore, a comparative study titled "Battle of the Large Language Models: Dolly vs LLaMA vs Vicuna vs Bard vs ChatGPT - A Text-to-SQL Parsing Comparison" [7] evaluates the performance of various LLMs in parsing Text-to-SQL. This work specially compares the gap between current open source and commercial models when it comes to text to SQL generation.

The integration of reasoning paired with action in LLMs for database interactions has also been analyzed. The paper in [8] introduces a dataset for long-form database question answering, challenging LLMs to generate and reason with multiple SQL queries. This study identifies key bottlenecks in planning and query generation and proposes a multi-agent evaluation framework to enhance the precision of evaluations.

When it comes to the realm of SQL equivalence reasoning, a study [9] presents a new framework based on U-semiring semantics, which improves the modeling of SQL features and enhances the capability of automated query equivalence checking. By using SMT solvers as the reasoning engine, this approach offers a more robust solution for real-world SQL queries.

Our work builds upon these previous studies by looking at the use of LLMs as SQL agents to facilitate user-database interactions. We leverage lightweight models such as Llama-2-7B and Mistral-7B to create an agent capable of producing accurate and concise SQL queries.

4 Data

Initial iterations of the project used OpenAI's API, however, due to restrictions within the AoU program on transferring private health information, a transition to a publicly accessible database was deemed necessary. The Cancer Genome Atlas (TCGA) was identified as an appropriate alternative among publicly accessible BigQuery databases. BigQuery databases are structured in such a way that the columns of each table can have extensive descriptions associated with them. Within the context of TCGA, the large number of tables coupled with the extensive, and somewhat redundant column descriptions resulted in over 50,000 tokens being generated from a simple query

of the TCGA database schema. The excessive number of tokens posed a challenge as the count would be too large for the smaller models, we ultimately aimed to implement in addition to being costly if implemented using OpenAI's API.

While extensive column metadata for the TCGA database was seemingly beneficial, it introduces complexities that were not fully appreciated at project outset. These challenges necessitated a simpler approach, thus we opted to use MySQL databases as an alternative. A local MYSQL database originally developed for the textbook Fundamentals of Database Systems by Elmasri and Navathe, and used in Georgia Tech's CS4400 course, appeared to be optimal for development purposes. Because the database was used for other course work, it also came with a large set of pre-made questions and their expected outputs. These expected outputs are incorporated into an "autograding" script, facilitating the evaluation of queries produced by our model. Additionally, many of the questions require queries that include multiple tables within the databases, parallel to the complexity we expect with queries to larger biological databases, such as AoU.

5 Methods

To tackle this problem of domain-specific generation, there were three main approaches that were taken: prompt engineering on powerful pre-trained, finetuning models, and a conjunction of two types of finetuning.

5.1 Prompt Engineering

As models continue to grow in power and training ability, many of the state-of-the-art models, such as Llama, GPT, and Mistral have immense prowess even in domain-specific fields, such as SQL generation. Prompt engineering methods, such as Chain-of-Thought, have shown promise especially in layered and more difficult tasks[6]. In terms of methods, we use Chain-of-Thought on Llama-2-7B and Mistral-7B. The reason we used Chain-of-Thought specifically is two-fold. The first is the propensity for more long-form tasks. When looking at the types of queries that are used in this specific domain, many of them are quite long-form and would most likely take multiple steps to manually come up with. These types of scenarios have shown great promise under Chain-of-Thought Prompting. Additionally, one of the benefits to SQL generation is the introduction of structure. SQL queries are very structured in how they are generated and have a very specific format, and Chain-of-Thought has shown promise in that realm. Additionally, there are two main benefits to integrating Prompt Engineering into a solution. The first is leveraging the power of pre-trained models. Companies like Meta, OpenAI, and more, have copious amounts of data and power, so leveraging this in our own scenarios is quite helpful. Additionally, the computational cost is quite minimal. By simply integrating changes within the actual prompt, the cost of generation becomes significantly lower, as no fine-tuning, training, or any computational cost other than loading and querying the model is incurred.

5.2 Finetuned Models

There are two types of finetuning that we implement. The first is instruction tuning. When doing qualitative analysis in the initial exploratory phase of the project, one part of the generation that language models struggled with was with the actual schema. Passing in the schema in an effective way for the language model to understand seemed to be one of the most pressing problems. To counter that, we implement instruction tuning, so that the models have a central way to understand

the schema. The general structure of the schema passed in was to have the first line be the name of the table, and each of the following lines to be each of the keys in a database, their type, and if they are a primary key, etc. The second type of finetuning was the traditional finetuning. Recently, Gretel AI had released the largest text-to-SQL dataset, where the dataset included the schema, general features of the dataset, the natural language description, and the actual SQL generation. Using this dataset, due to its diversity in types of models, language descriptions, and queries, we decided to use this specific dataset. For both types of finetuning, we used Llama-2 7B for finetuning, quantized to 4-bit, to help fulfill computational constraints. In terms of the actual finetuning process, we used qLORA, as full-finetuning and normal LORA presented problems computationally.

5.3 Multi-step Finetuning

The main model that is introduced is a combination of the above two finetuning models. The first step is to do base qLORA finetuning on the Gretel AI SQL dataset. That dataset's diversity allows for learning of the structure of SQL along with base understandings of the fastest ways to learn efficient commands. Having this as the first step allows the new updated model to get a more nuanced understanding of SQL itself. The main issue then lies in how to pass on more domain-specific knowledge and how to process the specific schema of the databases in our problem space. To attack that, we implement instruction tuning on the model after already finetuning it on the Gretel AI dataset. Having the model instruction tune over our specific schemas allows the model to be more nuanced and accurate in our problem space. Additionally, since the two different types of finetuning provide different benefits to the model, we posit that the benefits will compound when both methods are done in conjunction.

6 Experiments

In terms of evaluation, there is no singular metric that can capture the effectiveness of a language models' generation of a SQL query. The first thing to calculate is the actual generation accuracy. Generation accuracy simply just takes a generation and the actual SQL database that the model was querying. The generated query is then passed into the database, and if it succeeds, it counts as a successful query for this metric, if it does not generate an error. If the generation causes an error, due to a misspelling, incorrect SQL structure, etc., it counts as a failed query. The final metric is calculated as successful queries divided by total queries. This metric is important because we want to get a general idea of how good a model is at learning the general SQL structure based on the schema. Especially if the model is not as successful, this metric helps identify which parts of the problem that the model is failing at. The second main evaluation metric is the accuracy of the generation itself. Let's say we have a prompt and an expected return from the database. If the generated query when passed into the database results in the same return, it will be deemed a "successful" query. This pure accuracy is calculated as successful queries divided by the total number of queries. The final main metric that is calculated is adjusted query verbosity. When a natural language query is passed in, there are often many ways to generate a specific targeted dataset. These queries can be of varying lengths, and we want to make sure that the language model is not consistently extra verbose. With that, we come up with a metric, that is slightly based on BLEU, where we penalize longer queries. We use this exponential decay formula to have lower values for correct queries that are longer. If a query results in an incorrect response, it has a score of 0. This score is averaged for each model and compared, along with just the relative distribution.

The query penalization function $f(Q)$ is defined as:

$$f(Q) = \begin{cases} e^{\frac{-\text{Length of Generated Query}}{\text{Length of Ground-Truth Query}}+1} & \text{if query Q is correct} \\ 0 & \text{if query Q is incorrect} \end{cases}$$

7 Discussion

7.1 Quantitative Analysis

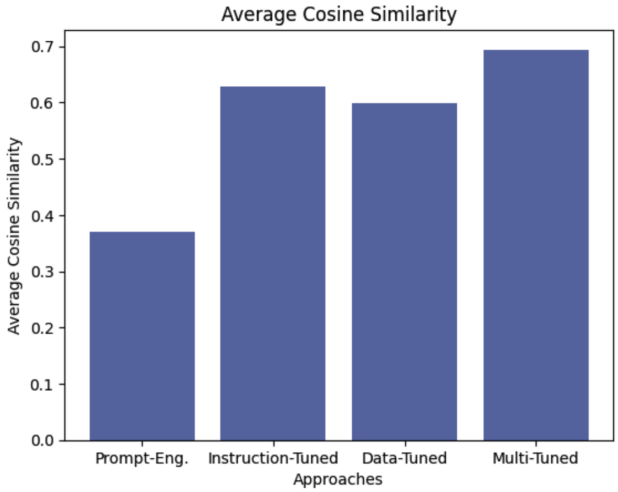

Figure 1: Cosine Similarity

7.1.1 Prompt Engineering

In terms of the quantitative analysis of prompt engineering, it was consistently the worst performing approach. It had the least cosine similarity between the response and the generated query. This indicates a lack of understanding of the prompt and SQL structure, comparatively. Additionally, the compilation accuracy and actual accuracy are quite low, indicating that the model does not have the greatest grasp on SQL itself. Especially with a low score on the compilation accuracy front, the model is not generating valid SQL queries. Since the model does not generate valid SQL queries most of the time, the query length penalization score does not have much impact, as it is quite low.

7.1.2 Instruction-Tuned Model

In terms of the instruction-tuning, there were varying levels of success for each different type of metric. There was a definite increase in the cosine similarity, indicating a more accurate model, generating more topically relevant queries. Additionally, the compilation accuracy saw a monumental increase. This increase indicated the instruction tuning profoundly impacting the knowledge attainment of the SQL structure in general. However, the impact is not profound on the actual accuracy and query penalization, indicating a lack of knowledge of the actual schema itself.

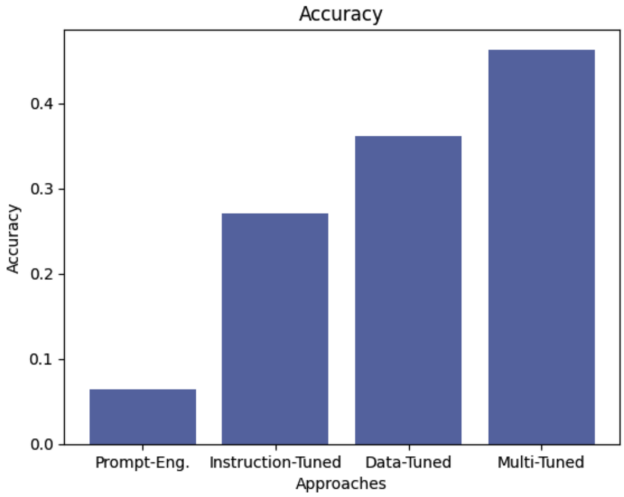

Figure 2: Accuracy

7.1.3 Dataset-Tuned Model

In terms of the dataset-tuned model, there was more profound impact on both the actual accuracy and query penalization. Both the accuracy and query penalization score saw sharp increases in this new model. This indicates that the model not only is getting a greater knowledge base in understanding SQL structure and queries, but also understanding the schema passed in. Additionally, the query penalization score indicates that the solutions that are generated by the model are either optimal or close to the optimal query to generate the resulting table.

7.1.4 Multi-Tuned Model

In terms of the multi-tuned solution, this was the most accurate model on all fronts. First, the calibration accuracy was higher than any of the other models, which is an indication of the new model's knowledge of SQL and generation of specific queries. Also, the query penalization score shows that all generations of this model's queries are close to optimal. This indicates that not only does the model have knowledge of the SQL structure and the database from the specific schema, but it also has knowledge of the optimal queries to generate specific tables and map to specific natural language queries.

7.2 Qualitative Analysis

7.2.1 Prompt Engineering

When investigating the outputs of the prompt engineered of both open-source models tested, the performance was much worse than the alternative methods. The responses were often filled with large explanations, discussing why the SQL query was outputted, and a step-by-step explanation. Additionally, even when the SQL query was extracted from the entire response, with the explanation removed, the metrics did not improve drastically. Qualitative analysis of some of the responses gave us two main observations regarding the model. The first is the lack of understanding of the database from its schema. The language model would often hallucinate and add extra keys, table names, etc. The second main observation is that the language model did not have a great understanding

of SQL itself. The model would often start queries with words other than SELECT, and would invent commands, that would invalidate a wide variety of SQL rules.

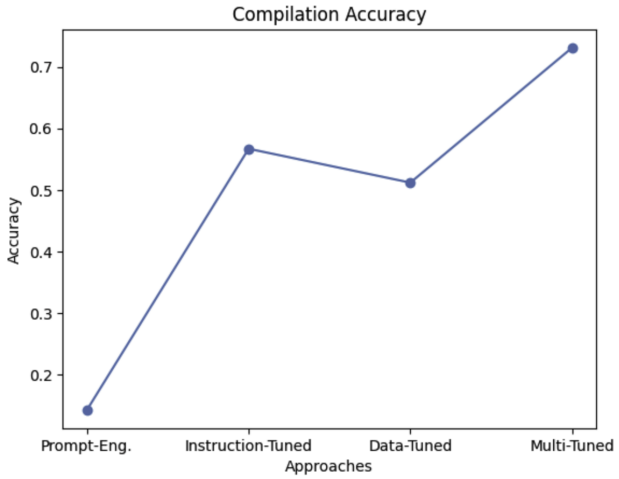

Figure 3: Compilation Accuracy

7.2.2 Instruction-Tuned Model

Instruction-Tuning showed measurable improvements over the baseline and prompt engineering approaches. The first main observation was the lack of explanation text or given text. When tuned in to instructions, the language models became a lot more used to returning queries that were consistent. Even when the language model would output queries that were wrong in theory, most of the issues would not be due to an apparent lack of understanding of SQL, but rather a lack of understanding of the schema itself.

7.2.3 Dataset-Tuned Model

Dataset-Tuning provided more benefits overall performance-wise compared to both instruction-tuning and prompt engineering. After finetuning the model with the dataset, the diversity of the dataset seems to be quite helpful. Qualitatively, the model's generations after tuning were a lot more effective on some of the more complicated tasks. The finetuned model looked to have a much greater grasp of what each of the databases had and how to do some of the aggregations and other complicated queries. Overall, it seemed to have a greater grasp of the SQL structure as well, which can be attributed to the diversity of the dataset.

7.2.4 Multi-Tuned Model

Multi-Tuning the model through two phases provided comprehensibly the best performance metrics. Qualitatively, looking through the generations of this fine-tuned model, it is indicative that this model has a grasp over both the general SQL structure and understanding the schema when it is passed in. A great indication of the model's performance is its ability to either be fully correct or comparatively quite close and the most difficult queries, that would often require multiple steps for a human annotator to decipher.

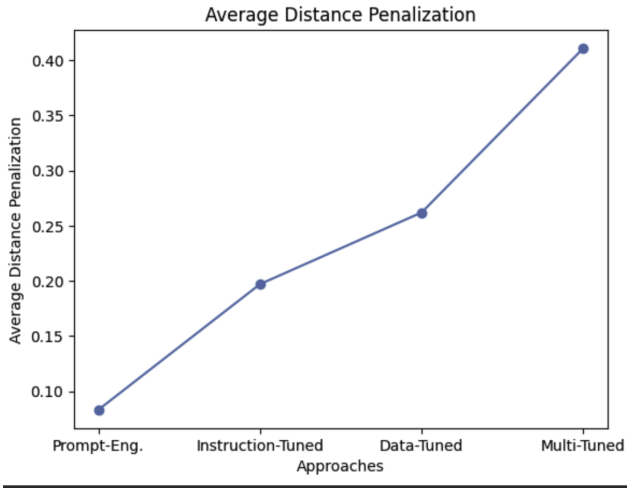

Figure 4: Distance Penalization

8 Conclusion

When looking at the specific issue of SQL generation, especially through the lens of domain-specific generations, there is a clear gap in what a human expert can do and what a language model can do. SQL generation is especially hard for two main reasons: an understanding of a new schema at every iteration and a general understanding of the SQL structure. At the beginning, we attempted to leverage some of the most powerful pre-trained models with possible prompt engineering strategies, such as Chain-of-Thought, but these methods did not seem to enhance the model's understanding. The team then attempted strategies that leveraged finetuning the model, both on the instruction front and through comprehensive natural language to SQL query datasets. Both types of tuning showed improvements in understanding SQL structure and the schema themselves. The novel solution that is introduced by the paper is this 2-pronged finetuning approach that finetunes the model on both instruction data and a pre-defined dataset to provide generations that are valid in SQL and can follow the natural language query. In the future, identifying specific types of schemas and using a Mixture-of-Experts approach with different types of databases and schemas could provide more optimal performance with marginal computational differences. Additionally, our approach was originally designed to be tested on biological databases, but due to time and privacy constraints, we were constrained to other pieces of data. For future work, we would continue to test this in the space of biology to test the effectiveness of our approach in a more domain-specific way.

9 Contributions

Raj: Worked on the instruction-tuning and finetuning of the model, while also implementing prompt engineering and all of the evaluation metrics. Additionally, worked on cleaning the dataset.

Colin: Worked on acquiring and initial querying of the TCGA BigQuery datasets and initial agent implementations, and also supplied and formatted the dataset used for testing.

Link to Github: https://github.com/Naughtoncolin/AoU$_{a}gent/tree/master$

10 References

[1] Gu, Zihui. Interleaving Pre-Trained Language Models and Large Language Models for Zero-Shot NL2SQL Generation, arxiv.org/pdf/2306.08891.pdf. Accessed 29 Apr. 2024.

[2] Zhang, Qinggang, et al. "Structure Guided Large Language Model for SQL Generation." arXiv.Org, 27 Mar. 2024, arxiv.org/abs/2402.13284.

[3] Touvron, Hugo, et al. "Llama 2: Open Foundation and Fine-Tuned Chat Models." arXiv.Org, 19 July 2023, arxiv.org/abs/2307.09288.

[4] Jiang, Albert Q., et al. "Mistral 7B." arXiv.Org, 10 Oct. 2023, arxiv.org/abs/2310.06825.

[5] Meyer, Yev. "Introducing World's Largest Synthetic Open-Source Text-to-SQL Dataset." RSS, Gretel.ai, 4 Apr. 2024, gretel.ai/blog/synthetic-text-to-sql-dataset.

[6] Wei, Jason, et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." arXiv.Org, 10 Jan. 2023, arxiv.org/abs/2201.11903.

[7] Querying Large Language Models with SQL, arxiv.org/pdf/2304.00472.pdf. Accessed 29 Apr. 2024.