DBMS Term Project Report

Team Heap Undercut

Atul Jayesh 21CS10012
Barun Parua 21CS10014
Owais Ahmad Lone 21CS10048
Ranjim Prabal Das 21CS10054
Navaneeth Shaji 21CS30032

15th April 2024

Contents

1 Introduction

This report details the development of our SQL query and results generation application, achieved through the fine-tuning of appropriate Large Language Models (LLMs) on SQL datasets. We provide a comprehensive overview of the steps taken throughout the development process, leading up to the final implementation. We also present the results and performance evaluation of our application in this document.

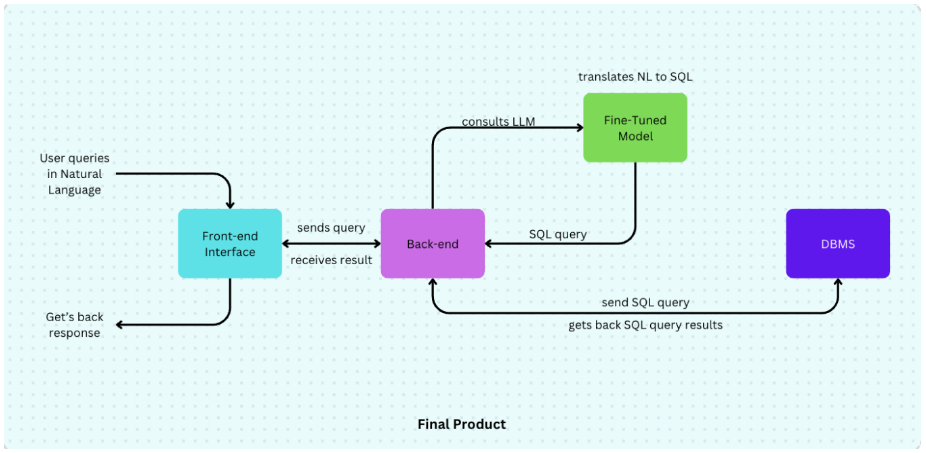
2 Objective

The primary objective of our Project "LLM SQL" was to train a general LLM with enough text queries and their corresponding SQL so that it can generate proper SQL queries on its own when given simple English Language Queries. As we know that LLMs are pretty good at generating Natural Language responses for our queries (for example, ChatGPT), we wanted to take that a step further to refine it to generate SQL queries accurately as well.

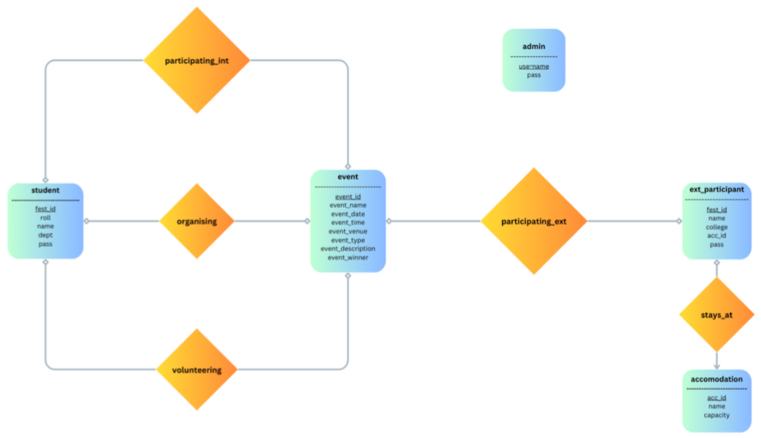
The primary goal of our project is to empower end-users to seamlessly access information from SQL-based database systems such as PostgreSQL, MySQL, and SQLite. They can get the proper SQL queries as well as the results from the database they are accessing provided they share the basic schema structure of the database and the natural language query regarding which they want to see the output. We know that nowadays most organizations use databases and a large fraction of them are SQL based. So this application makes the databases more accessible for end-users who might not be familiar with SQL syntax. Note that while our model is trained on just one database schema namely our Fest Database, with more powerful LLMs and more database specific queries, this application can be made more accessible to large real world databases like IRCTC or IIT ERP systems.

Additionally, our application features a mode that allows users to input queries without requiring the associated schema. While the generated SQL query may not always align perfectly with the context, it provides users with a foundational understanding of query construction. This mode serves as an educational tool, enabling users to familiarize themselves with SQL syntax through practical application.

2.1 Final Application Pipeline



2.2 ER Diagram of the Fest Database



3 Methodology

3.1 Basic Plan

In our methodology, we adopted a widely used approach for leveraging Large Language Models (LLMs). This involved fine-tuning the model for a specific task, considering its innate proficiency in understanding natural language. Note that even though an LLM is proficient in giving outputs for any general question, it still does not have a very good view of SQL syntax. Hence further fine-tuning is necessary.

Unlike freezing any part of the model parameters, we wanted to train as many parameters as possible to enhance its adaptability. Initially, we needed to train the model on a comprehensive text-to-SQL dataset lacking a specific schema. This allowed the model to grasp syntax intricacies. Subsequently, we refined its training on a dataset featuring a single schema, thereby enhancing its performance tailored to our specific requirements. This sequential training strategy ensured a more comprehensive understanding of SQL syntax and improved efficiency in generating accurate queries.

3.2 Datasets

Now, we had to look up various datasets which contain the SQL queries and the associated Natural Language statements. We found a couple of good datasets on HuggingFace and Github namely WikiSQL and Spider datasets. WikiSQL is a large dataset which has lots of queries and hence we used it for the first round of training in which the model basically learnt the SQL syntax. However, as the dataset does not contain schema along with the queries, we needed another round of training with schema specific examples. This is where Spider came in. It is a dataset that contained the schema metadata along with the queries that could be used for training. However the metadata information was in a very cumbersome JSON format which we further simplified into a CSV format from where the data pre-processing could be done much more easily.

Finally, we took our Fest Database from the previous assignment and generated hundreds of SQL queries and natural language questions on it. The reason we needed to do this is because we wanted to go even further and make the model give not just the relevant queries but also the associated outputs by executing the query on a PostgreSQL server.

3.3 Model

We used the FLAN-T5-Base model as our LLM for this project. It is a well known and popular LLM which is good at question answering tasks and requires relatively less number of parameters (248M params). It is an encoder-decoder model, with 12 encoder and 12 decoder layers. It has 248M parameters.

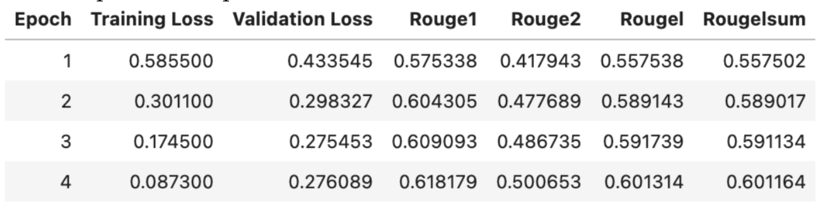
While we initially looked up a lot of different models like LLaMa, BERT, Gemini etc, most of them needed larger GPU memory than what Kaggle free tier provides and hence we could not do any effective training on them. Also as we got good enough results from the FLAN-T5 model itself, we kept using that only for our application.

4 Training and Results

Initially, we conducted fine-tuning on the WikiSQL dataset, comprising roughly 84k samples. We got a Rouge Score of 0.52 which indicated that the model had learnt some basic SQL structures. The model was trained for 4 epochs with the PyTorch trainer. The model is publicly available and can be accessed here. Note that Rouge Score is a well known metric used for evaluating automatic summarization and machine translation software in natural language processing.

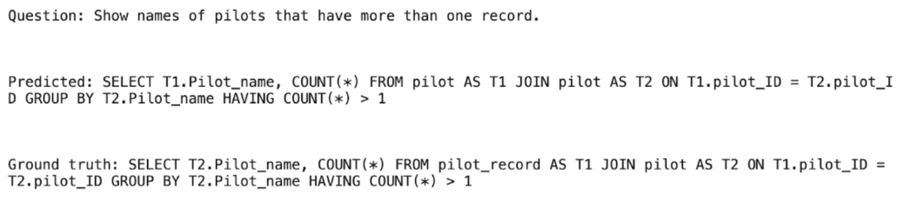
Now, the second round of training was carried out on the Spider dataset which had around 5k samples along with schema. We again trained using the PyTorch trainer and achieved a much better Rouge Score of 0.62.

After 4 epochs on Spider Dataset



Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum
1	0.585500	0.433545	0.575338	0.417943	0.557538	0.557502
2	0.301100	0.298327	0.604305	0.477689	0.589143	0.589017
3	0.174500	0.275453	0.609093	0.486735	0.591739	0.591134
4	0.087300	0.276089	0.618179	0.500653	0.601314	0.601164

Inferencing on the model

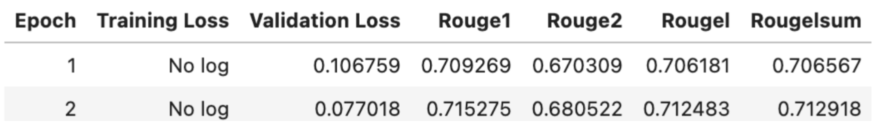


```
Question: Show names of pilots that have more than one record.


Predicted: SELECT T1.Pilot_name, COUNT(*) FROM pilot AS T1 JOIN pilot AS T2 ON T1.pilot_ID = T2.pilot_I
D GROUP BY T2.Pilot_name HAVING COUNT(*) > 1


Ground truth: SELECT T2.Pilot_name, COUNT(*) FROM pilot_record AS T1 JOIN pilot AS T2 ON T1.pilot_ID =
T2.pilot_ID GROUP BY T2.Pilot_name HAVING COUNT(*) > 1
```

The final round of training was done on our dataset of about 1000 handwritten queries based on our Fest Database. Due to the smaller number of queries, we trained the final model for just 2 epochs. Still, due to the well planned hierarchical way in which we trained the model, we got an excellent Rouge

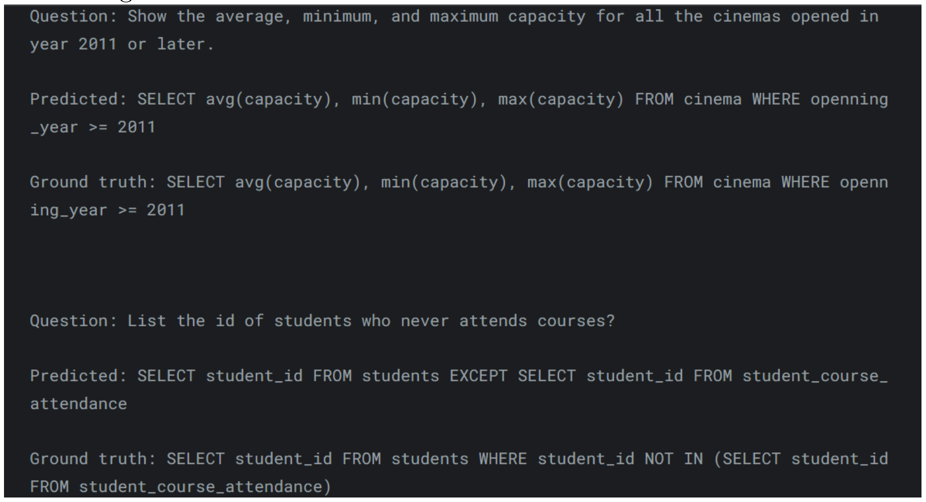
Score of 0.7 on the final model.

AFter 2 epochs on Fest Dataset



Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum
1	No log	0.106759	0.709269	0.670309	0.706181	0.706567
2	No log	0.077018	0.715275	0.680522	0.712483	0.712918

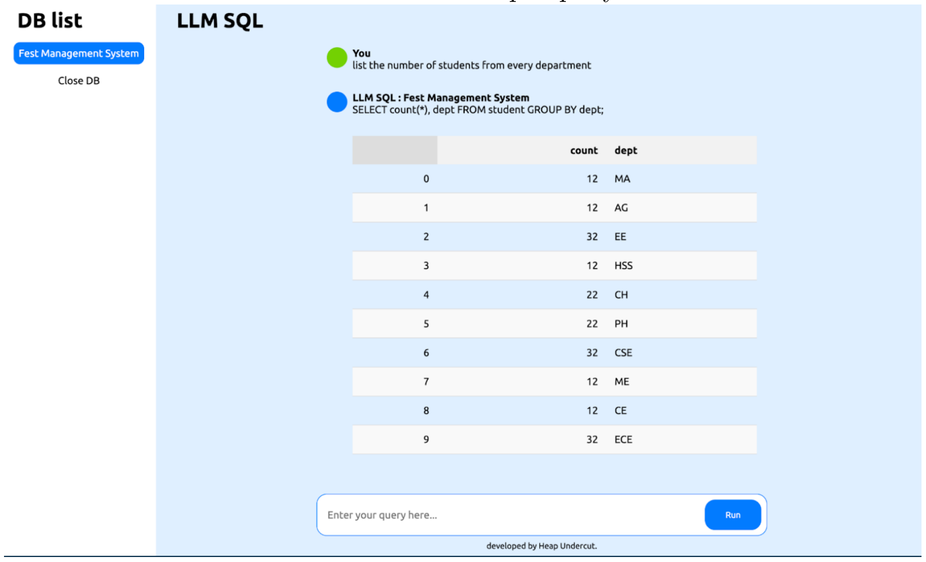
Inferencing on the model



```
Question: Show the average, minimum, and maximum capacity for all the cinemas opened in
year 2011 or later.

Predicted: SELECT avg(capacity), min(capacity), max(capacity) FROM cinema WHERE openning
_year >= 2011

Ground truth: SELECT avg(capacity), min(capacity), max(capacity) FROM cinema WHERE openn
ing_year >= 2011



Question: List the id of students who never attends courses?

Predicted: SELECT student_id FROM students EXCEPT SELECT student_id FROM student_course_
attendance

Ground truth: SELECT student_id FROM students WHERE student_id NOT IN (SELECT student_id
FROM student_course_attendance)
```

5 Interface Design

Basic UI of the Interface when asked a sample query



You
list the number of students from every department

LLM SQL : Fest Management System
SELECT count(*), dept FROM student GROUP BY dept;

	count	dept
0	12	MA
1	12	AG
2	32	EE
3	12	HSS
4	22	CH
5	22	PH
6	32	CSE
7	12	ME
8	12	CE
9	32	ECE

Next, we've developed a user-friendly website using Flask to interact with our SQL query generation model seamlessly. This website leverages the Hugging Face API to perform model inference without the need for complex model loading procedures. Users can choose between two modes of operation based on their data scenario.

In the structured mode, users select from preset schema options representing different data structures. They can then ask questions in plain English, and the model generates SQL queries tailored to the selected schema and user query. The query is then executed on whichever database the application is connected to (in our case a PostgreSQL server) and the query is returned in proper tabular form. This mode ensures accuracy and relevance in query generation, facilitating efficient data retrieval.

Alternatively, in the flexible mode, users can input queries in natural language without specifying a schema. The model, now interprets the queries and generates SQL queries accordingly. While this mode may offer less precision compared to the structured mode, it provides users with a more open-ended and adaptable approach to interacting with the model. Note that the user might give the schema in a structured way and then expect a proper response in this case as well.

Overall, our website, utilizing the Hugging Face API along with the psycopg2 connector, provides users with a straightforward and intuitive platform to communicate with our SQL query generation model. Whether users have a predefined schema or not, they can effortlessly articulate their data queries and receive SQL queries in response along with data if applicable, streamlining the process of data analysis and enhancing accessibility to SQL-based tools. This is much more easier and friendly than repeatedly running an IPython notebook somewhere else to simulate our model.

6 Conclusion

In this project, our central aim was to harness the capabilities of a general Large Language Model (LLM) to proficiently generate SQL queries with accuracy and relevance. To achieve this, we used two primary datasets: WikiSQL and Spider. These datasets provided us with a substantial corpus of general SQL queries, laying the groundwork for our model's training. Subsequently, we fine-tuned the model on our own database, refining its ability to generate SQL queries tailored to our specific schema.

However, despite our efforts, certain challenges emerged during the project. One notable challenge was the model's tendency to improvise attributes and table names when presented with improperly framed questions. This behavior occasionally led to inaccuracies in the generated queries, thereby impacting the output data. Additionally, we observed that the model's query outputs often deviated from conventional SQL syntax, further exacerbating the issue of obtaining proper data output. The model also sometimes gives quite different results if the similar queries are put up in different tones.

In conclusion, while our project encountered challenges along the way, it also signifies a significant step forward in the realm of natural language processing and SQL query generation. The progress we've made underscores the vast potential of LLMs in streamlining access to databases for users less familiar with SQL syntax. Moving forward, continued refinement and innovation in model training methodologies hold the promise of overcoming existing limitations and unlocking even greater utility and accuracy in generating SQL queries. With dedication and continued exploration, we remain optimistic about the future prospects of our project, envisioning a more seamless and accessible interface between users and database systems.

7 References

1. Our Repository
2. WikiSQL Dataset and Repository and Paper
3. Spider Dataset and modified CSV
4. Fest Database Dataset
5. Huggingface Inference Docs, Models and Datasets