SQL Chatbot for Databases

Zeynep TUTAR
zeynep.tutar@studenti.unipd.it

Mattia VARAGNOLO
mattia.varagnolo.1@studenti.unipd.it

Abstract

This project presents the development of a chatbot capable of converting natural language inputs into SQL queries, retrieving relevant information from a database, and returning results in a tabular format. The chatbot is built using a large language model (LLM), which was fine-tuned on a SQL dataset to enhance its ability to generate accurate SQL queries. Initial tests of the base model resulted in an accuracy of 20% with a similarity score of 0.9, indicating zero to minor deviations from the correct SQL query, for example using an alias instead of the table name, without impacting the correctness. After fine-tuning, the model's performance improved significantly, reaching above 44% accuracy at the same similarity score threshold.

The chatbot's performance was evaluated using a similarity-based metric, measuring the difference between the generated SQL queries and the ground truth queries. Further improvements could be made with more training epochs, as indicated by a decreasing loss function during validation. A Gradio interface was developed to make the chatbot accessible and user-friendly, allowing users to interact with the database through conversational input.

Despite the limitations posed by resource constraints, this project demonstrates the potential of fine-tuning LLMs for SQL query generation and natural language database interaction. Future work may include expanding the dataset, extending model training, and refining the chatbot's accuracy.

Keywords: Natural Language Processing (NLP), SQL Chatbot, Large Language Models (LLM), Query Similarity, Fine-tuning, Database Interaction, Llama 3.1, Gradio Interface.

1. Introduction

Interacting with databases using natural language queries presents a challenge for users without knowledge of SQL, which limits access to critical information for non-technical users. The goal of this project is to create an SQL chatbot that converts natural language inputs into SQL queries and retrieves relevant data from a database. This is achieved by fine-tuning a large language model (LLM) on SQL-specific tasks, enabling it to translate conversational queries into SQL commands.

This project aims to simplify querying and opening access to a broader range of users. The system is designed to process user inputs, generate SQL queries, and return results in a tabular format.

The following sections will detail the datasets and methodology used in the development and fine-tuning of the model, as well as the results obtained and the user interface.

2. Data

2.1. Datasets

For this project, we used two datasets: the gretelai/synthetic_text_to_sql dataset and the well-known Chinook SQLite database.

The gretelai/synthetic_text_to_sql dataset [3], sourced from Hugging Face, is a comprehensive and diverse collection of synthetic Text-to-SQL samples. The dataset includes 105,851 records, partitioned into 100,000 training and 5,851 test records. It covers approximately 23 million tokens, including 12 million SQL tokens, and spans across 100 distinct domains. This wide variety of SQL tasks includes data definition, retrieval, manipulation, analytics, reporting, and more complex operations such as subqueries, multiple joins, aggregations, window functions, and set operations. The detailed structure of the database is given with Figure 1. Each sample in the dataset is paired with a natural language explanation of the corresponding SQL query, making it an excellent resource for training models in Text-to-SQL tasks. We used this dataset for training and testing the base model.

Once the base model was fine-tuned, we tested the trained model using the Chinook database [4], which is a well-known SQLite database that mimics a digital media store. This allowed us to validate the model in a real-world database scenario. The Chinook database contains tables representing customers, orders, invoices, employees, and track information, as seen from Figure 2. The chatbot interface was specifically designed to interact with SQLite databases, ensuring seamless communication between the chatbot and this particular database format.

	column_name	Unique	Missing	Type	average token count	total token count
0	id	100000	0	int32	N/A	N/A
1	domain	100	0	object	2.205	233435
2	domain_description	100	0	object	18.229	1929598
3	sql_complexity	8	0	object	2.029	214809
4	sql_complexity_description	8	0	object	11.912	1260878
5	sql_task_type	4	0	object	2.882	305079
6	sql_task_type_description	4	0	object	9.737	1030683
7	sql_prompt	100338	0	object	17.098	1809835
8	sql_context	90034	0	object	82.136	8694143
9	sql	99605	0	object	30.578	3236692
10	sql_explanation	100113	0	object	44.868	4749329

Figure 1: Breakdown of gretelai/synthetic_text_to_sql dataset features and corresponding data types and token counts [3]

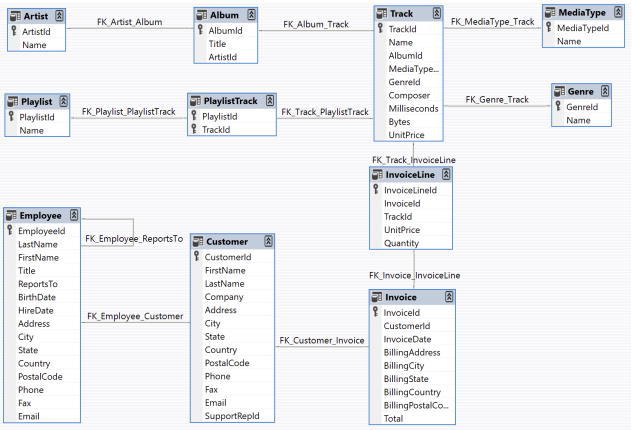

Figure 2: Chinook database structure

2.2. Preprocessing

Before training, we ensured that the gretelai/synthetic_text_to_sql dataset was clean. Preprocessing steps included removing any duplicate records, handling missing values, and ensuring consistency in the SQL syntax to prevent discrepancies during model training. Therefore, we decided to remove all the records unrelated to analytics and reporting, because we wanted our model to focus on SELECT queries and to become specialized on data gathering. This chatbot is not supposed to be able to write queries like INSERT or UPDATE, because they are risky operations which is not suggested to perform with a chatbot. These steps were necessary to ensure the dataset quality and maximize the model's learning potential. For testing, the Chinook database was already structured and formatted for immediate use in the chatbot interface.

3. Model Selection & Fine-Tuning

3.1. Model Selection

For this project, we selected the LLama 3.1 model [2] with 8 billion parameters as our base model. We also considered using the 70B version, but due to computational costs and memory constraints, we proceeded with the 8B model, which still offered a powerful and state-of-the-art performance. LLama 3.1 was chosen because it is one of the most advanced large language models currently available and is open-source, making it suitable for our research.

We applied 4-bit quantization to the model, a memory-efficient method that significantly reduces memory usage, allowing for faster downloads and better optimization of available resources. This quantization technique allowed us to fine-tune the model without running into memory overload issues.

3.2. LoRA Adapters

To further optimize the fine-tuning process, we incorporated LoRA (Low-Rank Adaptation) adapters, which reduce the number of parameters updated during training. This technique helps accelerate training by focusing on specific layers of the model, such as query and value projections, without sacrificing accuracy. The adapters were applied to specific layers (e.g., q_proj, v_proj, etc.) with a rank of 256 and no dropout to ensure the model could handle large-scale SQL queries effectively. We tried different configurations and dimensions, but after 256 we ran out of memory. However, experimentations confirms that higher values were able to produce better results.

3.3. Prompting

We created a custom prompt format that included an instruction, input context (i.e., database schema), and the desired SQL query as a response. The dataset was preprocessed to focus on the "analytics and reporting" SQL tasks to provide the model with a well-defined scope during training.

During training, we implemented early stopping with a patience of 10 steps to prevent overfitting or useless training if the model wasn't improving in general. We monitored evaluation loss throughout the process to ensure that the model was learning effectively.

3.4. Baseline Performance & Fine-Tuning

Initially, the model was evaluated without any fine-tuning to establish a baseline performance. Using the gretelai/synthetic_text_to_sql dataset, we tested the base model's ability to generate SQL queries based on natural language inputs. At this stage, the base model achieved a **20%** accuracy with a similarity score of **0.9**, indicating that while

the model had some understanding of SQL, it required improvement to handle more complex queries.

The fine-tuning process involved a training setup that used the SFT (Supervised Fine-Tuning) Trainer from the Hugging Face transformers library. We ran the training with a batch size of 2 per device and accumulated gradients over 4 steps to manage memory usage effectively. The total number of training steps was limited to 300, with an initial learning rate of 2e-4. We utilized mixed-precision training (FP16) to further optimize memory usage.

The evaluation was conducted every 10 steps, and the model checkpoints were saved every 60 steps to track performance over time. The primary metric used to select the best model was evaluation loss, and we applied early stopping based on this metric to avoid unnecessary training when improvements flattened out.

4. Experiments & Performance Evaluation

We evaluated the fine-tuned model on the test subset of the gretelai/synthetic_text_to_sql dataset, using a similarity score metric to compare the generated SQL queries with the ground truth. The similarity was calculated using the `difflib` library to assess how closely the generated SQL matched the correct query. We set three different thresholds, considering how much accuracy we wanted to achieve.

- For a similarity threshold of **0.8**, if a query has 0.83 similarity score for example, the SQL query generated is partially correct, sometimes just missing a column inside the SELECT statement. In this case, we reached around 54% of accuracy instead of 30% on baseline.
- For a similarity threshold of **0.9**, the SQL query generated is correct, and the "mismatch" is mostly due to different sintax usage, like using an alias (FROM table AS A) instead of the table name (FROM table). In this case, we reached around 44% of accuracy, representing a 24% improvement over the baseline score.
- For a similarity threshold of **0.95**, the SQL query generated is completely correct. In this case, we reached around 33% of accuracy, representing a good over the baseline score, highlighting the success of the fine-tuning process in refining the model's SQL generation capabilities.

Model	80% Similarity	90% Similarity	95% Similarity
Baseline Model Accuracy	30.67%	20.00%	17.00%
Trained Model Accuracy	58.00%	42.33%	33.00%

Table 1: Model Accuracy Comparison

5. Chatbot User Interface

We developed the SQL chatbot user interface using Gradio [1], an open-source tool that allows for building customizable interfaces for machine learning models. This interface enables to input natural language questions, which are then processed by our fine-tuned model to generate SQL queries. These queries are executed on a connected SQLite database, and the results are returned as a respose in tabular format.

5.1. Interface Functionality

The Gradio interface includes several key features that make interacting with the model and database more natural:

- **Model Loading:** We provided users the option to load the fine-tuned LLaMA 3 (8B) model through a dropdown menu. The model is loaded using the `FastLanguageModel.from_pretrained` function from the UnSloth library, which initializes both the model and tokenizer.
- **Database Connection:** We designed the interface to allow users to connect to a SQLite database by providing a URL to an SQL script. The script is downloaded and executed in an in-memory SQLite database, enabling dynamic querying. We retrieve the database schema for use as context when generating SQL queries.
- **Query Generation:** After the user inputs a question, the model generates an SQL query by incorporating the provided schema. This query is executed against the database, and the results are displayed in a table. We implemented prompt templates to format the input and output, making it easier for the model to understand and generate the appropriate queries.
- **Error Handling:** In case of errors, such as invalid queries or failed database connections, we built error messages that help users troubleshoot and correct the issue.

5.2. How the Chatbot Works

The workflow of the chatbot can be summarized as follows:

1. First, we retrieve the database schema, allowing the model to understand the structure of the connected database.
2. Using this schema and the user's natural language question, we create a custom prompt and pass it to the fine-tuned LLama 3.1 8B model.

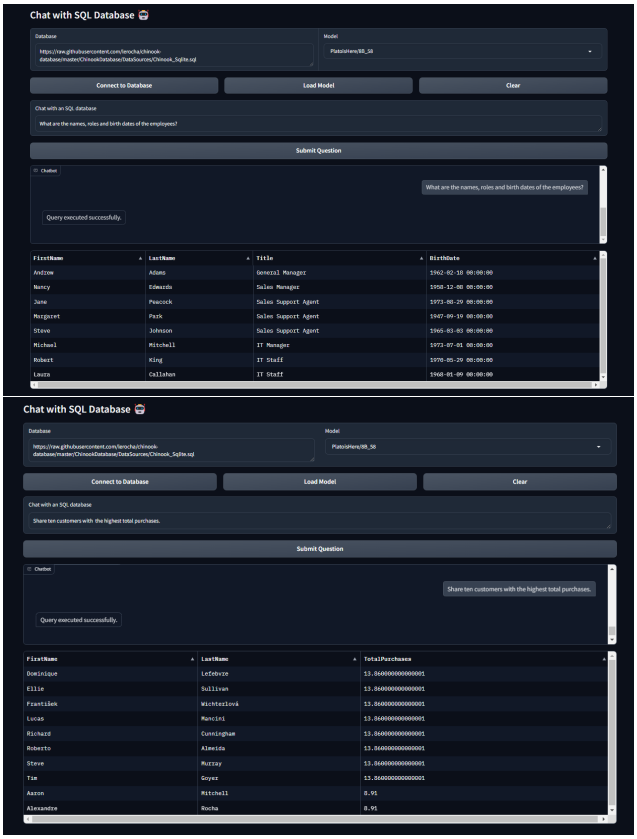

Figure 3: Gradio UI examples

3. The model processes this input and generates the appropriate SQL query, which is then executed through the SQLite database.

4. Finally, the results of the query are converted into a dataframe which is correctly showed to the user.

The interface includes features like a Clear button to reset the chat history and inputs, and a Submit button to process the user query and retrieve the results.

5.3. Example Use Case

For example, when a user asks, ”What are the names of all employees?”, the system retrieves the schema to identify the relevant tables, generates an SQL query like SELECT name FROM Employee, and executes this query. The results are showed as a dataframe inside the Gradio interface. Examples are shared in Figure 3

This interface allows users with no SQL knowledge to query the database conversationally, providing a more intuitive and accessible way to retrieve data from structured databases.

References

[1] Abubakar Abid, Ali Abdalla, Ali Abid, Dawood Khan, Abdulrahman Alfozan, and James Zou. Gradio: Hassle-free sharing and testing of ML models in the wild. https://arxiv.org/abs/1906.02569,, June 2019.

[2] Meta AI. Meta Llama 3.1. Accessed from: https://ollama.com/library/llama3.1, 2024.

[3] Yev Meyer, Marjan Emadi, Dhruv Nathawani, Lipika Ramaswamy, Kendrick Boyd, Maarten Van Segbroeck, Matthew Grossman, Piotr Mlocek, and Drew Newberry. Synthetic-Text-To-SQL: A synthetic dataset for training language models to generate sql queries from natural language prompts, April 2024.

[4] Luis Rocha. Chinook SQLite Database. Accessed from: https://raw.githubusercontent.com/lerocha/chinook-database/master/ChinookDatabase/DataSources/Chinook_Sqlite.sql, 2008.

Project Work Plan

Project Planning and Setup

- **Step 1: Define Project Scope and Objectives**
  - **Action:** Wrote a project work plan document outlined the project goals, scope, deliverables, and timelines.
  - **Deliverable:** Project Work Plan Document
  - **Completed By:** Zeynep TUTAR (1 hr) & Mattia VARAGNOLO (1 hr)
- **Step 2: Set Up Project Infrastructure**
  - **Action:** Set up a version control system on GitHub and final report template on Overleaf.
  - **Deliverable:** Project repository and report template.
  - **Completed By:** Zeynep TUTAR (1 hr)

Data Collection and Preparation

- **Step 3: Identify and Collect SQL Dataset**
  - **Action:** Researched and collected SQL dataset from GitHub and HuggingFace.
  - **Deliverable:** A SQL dataset for training and another for experimenting.
  - **Completed By:** Zeynep TUTAR (4 hrs) & Mattia VARAGNOLO (4 hrs)
- **Step 4: Data Cleaning and Preprocessing**
  - **Action:** Cleaned and preprocessed the datasets to ensure usability for training and testing.
  - **Deliverable:** Cleaned and preprocessed SQL datasets.
  - **Completed By:** Zeynep TUTAR (2 hrs) & Mattia VARAGNOLO (2 hrs)

Model Selection and Baseline Creation

- **Step 5: Select Pre-trained LLM**
  - **Action:** Chose a suitable pre-trained LLM that will be fine-tuned for SQL understanding.
  - **Deliverable:** Llama 3.1 8B model is selected.
  - **Completed By:** Zeynep TUTAR (4 hrs)
- **Step 6: Establish Baseline Performance**
  - **Action:** Tested the selected LLM on the SQL dataset before training for baseline performance measures.
  - **Deliverable:** Baseline performance outputs.
  - **Completed By:** Mattia VARAGNOLO (4 hrs) & Zeynep TUTAR (2 hrs)

Fine-tuning the LLMs

- **Step 7: Create a Pipeline**
  - **Action:** Created the training pipeline and the data manipulation processes.
  - **Deliverable:** Training and evaluation code.
  - **Completed By:** Mattia VARAGNOLO (6 hrs)
- **Step 8: Fine-tune LLM with SQL Dataset**
  - **Action:** Fine-tuned the selected LLM using the SQL dataset. .
  - **Deliverable:** Fine-tuned LLMs.

  - **Completed By:** Zeynep TUTAR (12 hrs) & Mattia VARAGNOLO (12 hrs)

- **Step 9: Validate Fine-tuned Model**
  - **Action:** Validated the performance of the fine-tuned models using a separate test set to ensure the models are learning correctly.
  - **Deliverable:** Validation performance outputs.
  - **Completed By:** Mattia VARAGNOLO (7 hrs)

Testing and Evaluation

- **Step 10: Test Model Performance**
  - **Action:** Conducted tests on the fine-tuned model to ensure it correctly interpreted natural language inputs and generated appropriate SQL queries.
  - **Deliverable:** Tested performance outputs with identified issues and fixes.
  - **Completed By:** Zeynep TUTAR (6 hrs)

- **Step 11: Evaluate Model Performance**
  - **Action:** Evaluated the overall performance of the fine-tuned model using accuracy and similarity.
  - **Deliverable:** Overall model evaluation.
  - **Completed By:** Mattia VARAGNOLO (10 hrs)

- **Step 12: User Interface**
  - **Action:** Created a Gradio user interface for better interaction.
  - **Deliverable:** Project documentation and presentation.
  - **Completed By:** Zeynep TUTAR (8 hrs)

Documentation

- **Step 13: Document the Project**
  - **Action:** Prepared comprehensive documentation covering all aspects of the project, including data collection, model fine-tuning, testing, and evaluation.
  - **Deliverable:** Project documentation.
  - **Completed By:** Zeynep TUTAR (8 hrs) & Mattia VARAGNOLO (2 hrs)

Total Project Hours

- **Zeynep TUTAR:** 48 hours
- **Mattia VARAGNOLO:** 48 hours