NLP SQL CHATBOT

USING GEMINI PRO

CISC 6210 Natural Language Processing Final Project Presentation

Presented by: Sarah Al Jamal and Deepamol Chacko

Date: 10th May, 2024

Fordham University, New York

Overview

- Traditional database querying requires knowledge of structured query languages such as SQL, which can be a barrier for many users. This limits their ability to access, explore,and analyze complex datasets, hindering decision-making and insights.

- The NLP SQL Chatbot aims to remove this obstacle by translating natural language queries into SQL, allowing users to interact with databases seamlessly and intuitively.

Purpose

To develop an application that translates natural language queries into executable SQL commands to retrieve data from a database efficiently.

Goals

- Enable users to interact with databases without knowledge of SQL.
- Streamline data access using natural language processing instantly.

Introduction

Background: The increasing need for intuitive data retrieval methods in software applications has led to the development of this project.

Relevance: This tool lowers the barrier for non-technical users to interact with complex databases.

Tools and Techniques Used

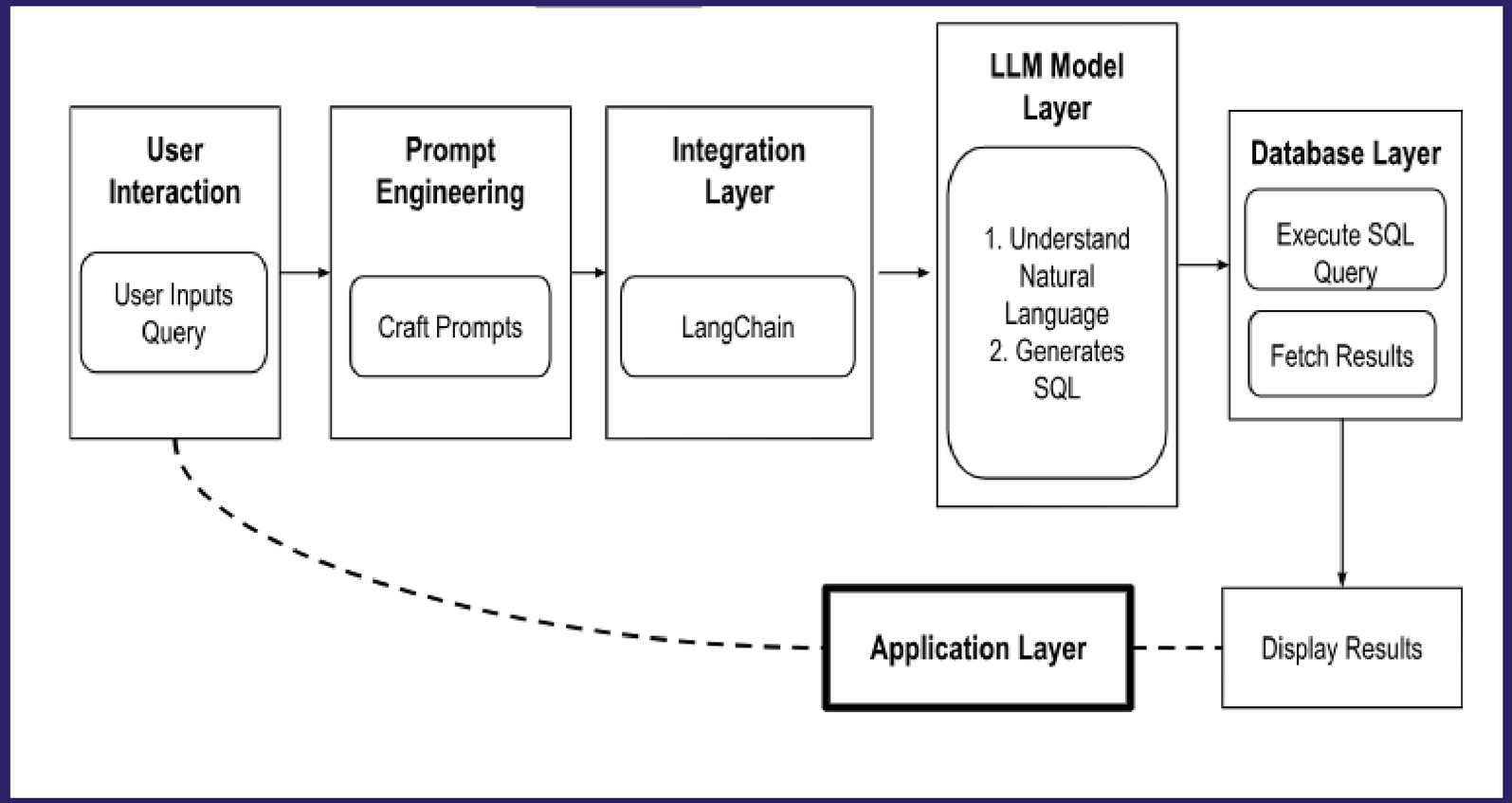
- Programming Languages: Python.
- Libraries: Google Generative AI for NLP processing.
- Web/Application interface : **Streamlit**
- Database management : SQLite3
- Requests for API interactions.
- APIs: https://aistudio.google.com/app/apikey

```
cursor.execute('''
    CREATE TABLE IF NOT EXISTS COURSES(
        Course_Id INT,
        Course_Name VARCHAR(50),
        Prof_Name VARCHAR(50),
        Stud_Count INT
    );
''')

print("Inserting records into STUDENT table...")
# Insert statements for STUDENT table
cursor.execute("INSERT INTO STUDENT VALUES ('Sarah', 'AlJamal', 'Natural Language Processing', 1002)")
cursor.execute("INSERT INTO STUDENT VALUES ('Lara', 'Sharaby', 'Algorithms', 1003)")
cursor.execute("Insert Into STUDENT values('Deepa', 'Chacko', 'Natural Language Processing', 1002)")
cursor.execute("Insert Into STUDENT values('Malak','Mohammed','Cloud Computing', 1004 )")
cursor.execute("Insert Into STUDENT values('Ali', 'Gahmy', 'Data Mining', 1005)")
cursor.execute("Insert Into STUDENT values('Sara', 'Wilson', 'Machine Learning', 1006)")
cursor.execute("Insert Into STUDENT values('Rohan', 'Kumar', 'Machine learning', 1006)")
cursor.execute("Insert Into STUDENT values('Pat', 'Li', 'Data Mining', 1005)")
cursor.execute("Insert Into STUDENT values('Rachel', 'Edwards', 'Deep Learning', 1007)")
cursor.execute("Insert Into STUDENT values('Mohammed', 'Ali', 'Algorithms', 1003)")
# Additional inserts here if needed

print("Inserting records into COURSES table...")
# Insert statements for COURSES table
cursor.execute("INSERT INTO COURSES VALUES (1002, 'Natural Language Processing', 'Dr. SpencerLuo', 20)")
cursor.execute("INSERT INTO COURSES VALUES (1003, 'Algorithms', 'Dr. Wenqi', 22)")
cursor.execute("INSERT INTO COURSES VALUES (1004, 'Cloud Computing', 'Prof. Zhou', 22)")
cursor.execute("INSERT INTO COURSES VALUES (1005, 'Data Mining', 'Dr. Yijun', 24)")
cursor.execute("INSERT INTO COURSES VALUES (1006, 'Machine Learning', 'Dr. Ruhul', 30)")
cursor.execute("INSERT INTO COURSES VALUES (1007, 'Deep Learning', 'Dr. Yijun', 20)")
# Additional inserts here...
```

Data Set

Customized the Dataset: Created Two Tables STUDENT and COURSES and inserted data manually.

Model Architecture



- User Interaction: Text input through Streamlit.
- LLM Model Layer: Gemini Pro API to process and translate text.
- Database Layer: Executes SQL on SQLite and returns results.
- Application Layer: Streamlit for displaying results.
- Explanation: "The architecture supports translating user inputs via an NLP model, executing SQL in a database, and presenting results through a web interface."

Question Preprocessing

```python
# Configure our API key
genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))

def preprocess_query(question):
    question = question.strip().lower()
    question = re.sub(r'[^\w\s]', '', question)
    spell = SpellChecker()
    words = question.split()
    corrected_words = [spell.correction(word) for word in words]
    corrected_question = ' '.join(corrected_words)
    replacements = {'entries': 'records', 'students': 'pupils'}
    corrected_question = ' '.join(replacements.get(word, word) for word in corrected_question.split())
    return corrected_question
```

This function preprocesses a text input by cleaning, standardizing, spell-checking, and applying specific word replacements to make it ready for further use, potentially for tasks like querying a database or performing natural language processing.

```
# Define the Prompt
prompt = [
    """
    As an SQL expert, you transform English questions into precise SQL queries.
There are two tables in the SQL database: STUDENT and COURSES.
The STUDENT table has the columns First_Name, Last_Name, Course_Name, and Course_Id.
The COURSES table includes Course_Id, Course_Name, Prof_Name, and Stud_Count.

Here are examples of how you should convert questions into SQL queries:

Example 1: "How many students are there in total?" the SQL command will be something like this:
"SELECT COUNT(*) FROM STUDENT;"

Example 2: "Who are all the students enrolled in the Natural Language Processing class?" the SQL command will be something like this:
"SELECT First_Name, Last_Name FROM STUDENT WHERE Course_Name = 'Natural Language Processing';"

Example 3: "What is the name of the professor teaching Cloud Computing?" the SQL command will be something like this:
"SELECT Prof_Name FROM COURSES WHERE Course_Name = 'Cloud Computing';"

Example 4: "How many students are taking the Algorithms course?" the SQL command will be something like this:
"SELECT Stud_Count FROM COURSES WHERE Course_Name = 'Algorithms';"

Example 5: "List all courses along with their respective professor names." the SQL command will be something like this:
"SELECT Course_Name, Prof_Name FROM COURSES;"

Example 6: "Find the names of students enrolled in Machine Learning courses." the SQL command will be something like this:
"SELECT First_Name, Last_Name FROM STUDENT WHERE Course_Name = 'Machine Learning';"

Example 7: "Which course has the highest number of students?" the SQL command will be something like this:
"SELECT Course_Name, MAX(Stud_Count) FROM COURSES;"

Example 8: "Display all information for students named 'Sarah'." the SQL command will be something like this:
"SELECT * FROM STUDENT WHERE First_Name = 'Sarah';"

Example 9: "Count the number of courses each professor is teaching." the SQL command will be something like this:
"SELECT Prof_Name, COUNT(Course_Id) AS Num_Courses FROM COURSES GROUP BY Prof_Name;"

Example 10: "Who teaches the course with the lowest student count?" the SQL command will be something like this:
"SELECT Prof_Name FROM COURSES WHERE Stud_Count = (SELECT MIN(Stud_Count) FROM COURSES);"

The two tables STUDENT and COURSES are joined using the primary key Course_Id. Please create the table relationship if necessary.

Make sure the SQL queries are clean and direct, avoiding any unnecessary characters such as triple quotes or the keyword 'sql' in the output.
    """
]
```

Prompt Engineering

Manually added all possible queries and prompts which created a comprehensive prompt to guide the model

Langchain

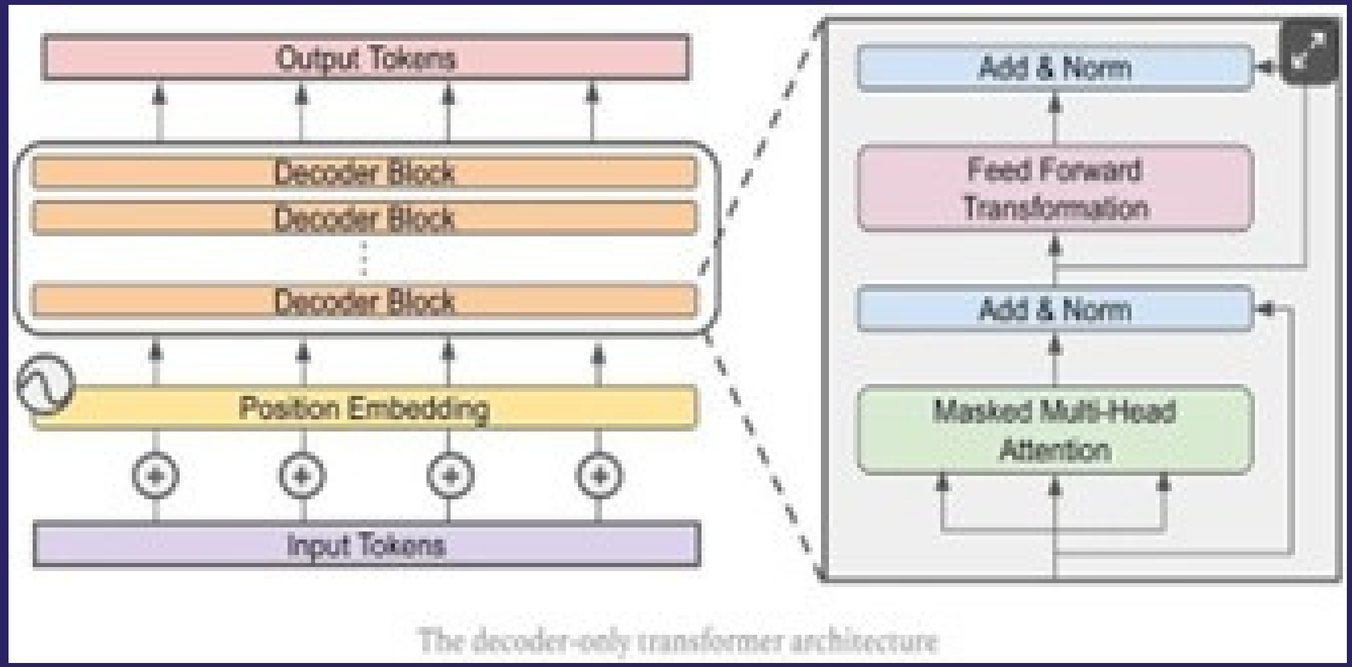
What is a langchain: Langchain is a tool that helps language models, like chatbots, to connect with external databases and other online resources. It makes these models smarter and more helpful by giving them access to a lot more information, helping them act more like humans in their responses.

LangChain Integration with LLM Model: LangChain was configured to interface seamlessly with both the LLM and the SQLite database, ensuring that SQL queriesare generated accurately and efficiently.

Gemini Pro Model

For our project, to understand the natural language queries, we are using the Gemini model, developed by Google, which is a powerful generative AI that has been trained on a diverse dataset of textual data, providing it with the ability to understand and generate natural language content effectively, that, likely encompasses a broad range of sources,including websites, books, articles, and other publicly available texts, giving the model exposure to a variety of linguistic patterns, terminologies, and contexts.

The Gemini model builds on the Transformer decoder only architecture, leveraging its powerfulprocessing capabilities for NLP tasks



The decoder-only transformer architecture

CHALLENGES

Integration Issues with LangChain:

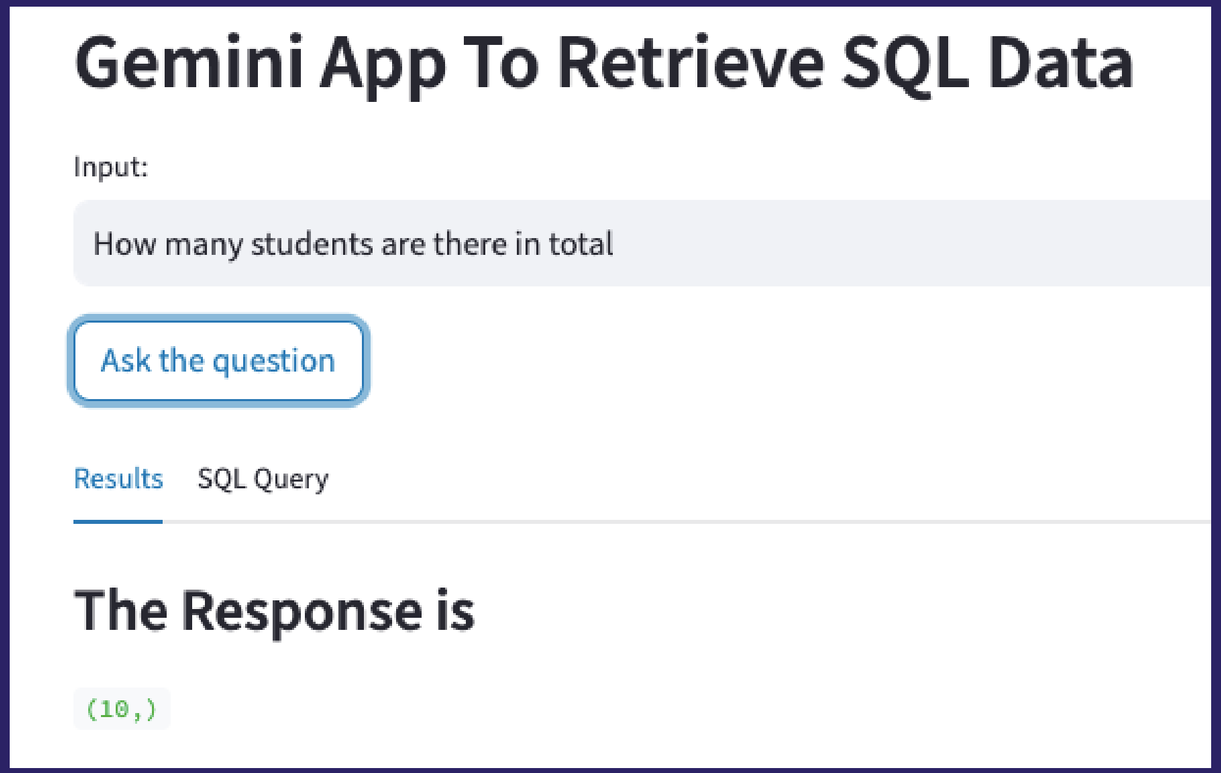
Initial attempts to integrate LangChain for SQL translation were unsuccessful due to compatibility issues and library limitations in case of Gemini Pro.

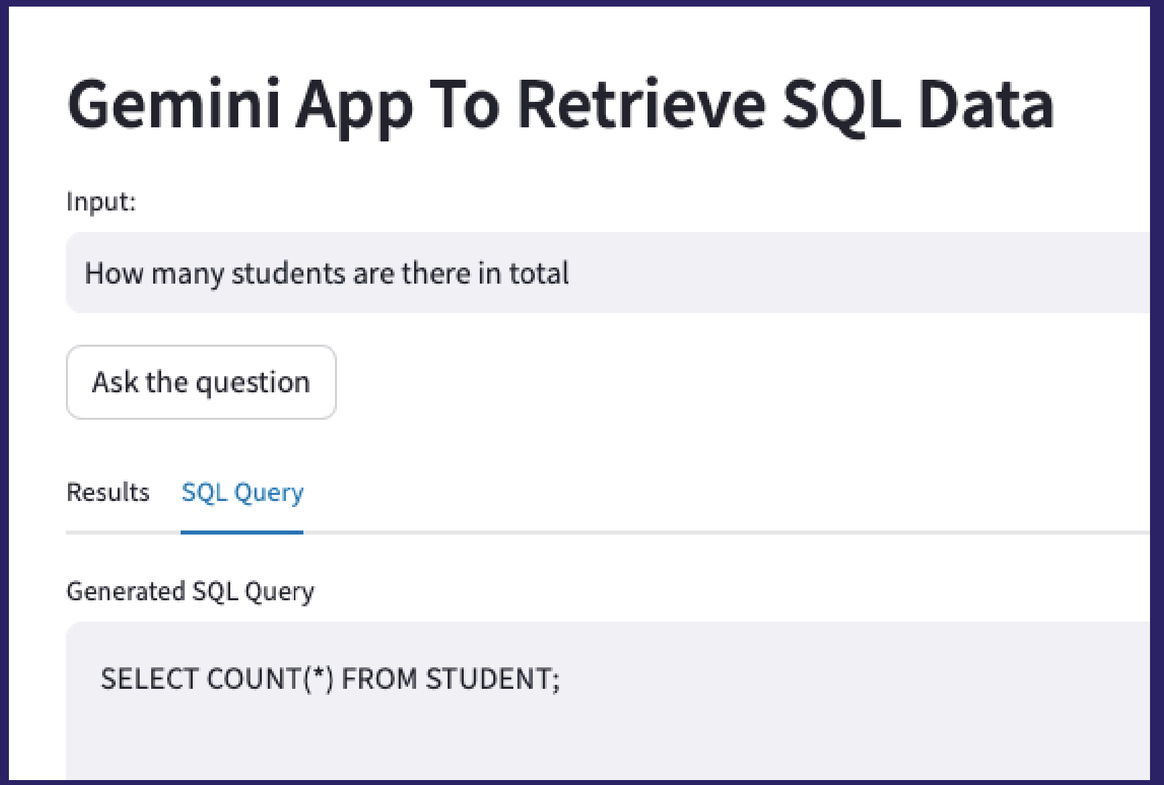
API Response Handling:

Challenges in handling and interpreting JSON responses from the Gemini Pro API.

RESULTS

The application successfully translates user questions into SQL codes and return the response like 'total students in the student table' into correct SQL queries.





Demonstration:

Live demo of the app in action, showing both the query and the resulting data.

Conclusion

Achievements:
Successfully bypassed the need for direct SQL knowledge for end-users. Created a robust application that utilizes advanced NLP capabilities to translate user queries into SQL, enhancing interaction with databases without needing SQL knowledge. This setup ensures the application is user-friendly and leverages modern AI technology effectively.

Learning Points:
Gained insights into NLP's application in SQL query generation and practical API usage.

Future Enhancements:

- Plan to add more complex query handling and support for additional databases.
- Using memory with Chatbot

THANK YOU