NLP SQL CHATBOT

Sarah Al Jamal
Fordham University
saljamal@fordham.edu

Deepamol Chacko
Fordham University
dchacko1@fordham.edu

Abstract– **This report discusses the development of an NLP SQL Chatbot that utilizes LangChain and Streamlit to enable users to interact with a database using natural language queries. The chatbot translates these queries into SQL commands, facilitating an intuitive user-interface experience without requiring SQL knowledge. The project demonstrates successful integration of advanced NLP techniques with database management systems, offering a practical solution for user-friendly database querying. Leveraging a combination of pretrained Google gemini pro model along with LangChain integration, this chatbot demonstrates significant improvements in query accuracy and efficiency. We trained the model using the demo data that we have created with query pairs spanning multiple domains, ensuring robustness and wide applicability. Our evaluations, based on testing against existing benchmarks, show that this chatbot achieves accuracy rate of (the exact evaluation is still under progress). The system not only enhances user interaction with relational databases but also offers insights into improving semantic parsing techniques for complex query generation. Our work contributes to bridging the gap between conversational language understanding and database querying, marking a step forward in user-centric data access technologies."**

Keywords: Natural language processing(NLP), LangChain, Prompt Engineering, Large Language Model (LLM), SQLite

I. Introduction

In today's world, being able to quickly get and understand data is very important. SQL databases, where lots of data is kept, are not easy for everyone to use because you need to know special programming commands. Natural Language Processing, or NLP, helps solve this problem. NLP lets people ask questions and give commands in everyday language, just like talking to another person, to get information from these databases.

NLP makes it possible for more people to use complex data systems easily. It allows people who are not programmers to ask questions and get answers from databases without needing to learn programming. This makes it easier for everyone to use the data they need.

In the domain of database management and getting relevant data from the source database optimally, the technologist has to be an expert in doing so. Even though an expert can utilize the queries and display the results, still they are facing problems with fetching relevant tables or data from the database, which is also very time consuming and sometimes errors might occur. Especially, in this digital age, accessing and analyzing efficient data are prime factors for decision making across all industries. While SQL databases remain central repositories of structured data, it requires specific syntax knowledge, which can be a barrier for non-technical users. In this scenario, the ability to convert natural language queries into SQL commands simplifies interactions for users without technical expertise. This paper presents an innovative NLP SQL chatbot, designed to facilitate seamless translation of user-generated natural language into precise SQL statements. The implementation demonstrates robust query handling, with StreamLit providing a dynamic user interface.

Objective

The main goal of this project is to create a chatbot that understands questions asked in everyday language and finds the answers by turning these questions into SQL commands. We will use something called LangChain to help the chatbot understand and create these commands because

LangChain works well with big, advanced programs that understand language.

We also want to make sure that the chatbot is easy for people to use, so we are using a tool called Streamlit to build a friendly interface. This interface will let people type their questions, show them the answers, and help them if they need to ask better questions.

This chatbot is designed to do two main things:

1. **Make Data Easy to Reach:** It will help people who are not tech-savvy to easily get data insights without needing to know programming.
2. **Save Time for Everyone:** It will also help those who already know programming to get their data faster, so they can spend more time on other important tasks.

The end goal is to make it simpler for all kinds of users to access and use data, which can help them make better decisions and be more productive.

Problem Statement:

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. This solution seeks to democratize data access, enabling individuals from diverse backgrounds to unlock valuable insights from structured data repositories without requiring specialized technical skills.

II. Related work

Overview

This part of the introduction talks about other projects and studies that are similar to our chatbot project. It helps us understand what has been done before and how our project fits into these previous works.

Natural Language Interfaces to Databases (NLIDB):
Early attempts to create interfaces that allow users to interact with databases using natural language. Examples include systems developed in the 1970s and 1980s like LUNAR, which allowed scientists to query lunar rock data using natural language. These systems laid the foundational ideas for using NLP in database querying, demonstrating the potential to simplify complex data interactions.

Recent Advances in NLP:
The introduction of models like OpenAI's GPT and BERT by Google, which have significantly advanced the understanding and generation of natural language by computers. These models have enhanced the accuracy and flexibility of NLIDBs, allowing more complex and varied queries to be understood and executed.

Chatbot Technology for Business Applications:
The use of chatbots in customer service to handle inquiries through natural language, as seen in platforms like IBM Watson Assistant. Demonstrates the practical application of NLP in real-world scenarios, providing insights into user interaction patterns and system integration challenges.

Integration of NLP with SQL Databases:
Projects and tools that specifically focus on translating natural language queries into SQL, such as SQLBot or the research done on adapting large language models for specific applications. Directly related to the current project, these tools illustrate specific challenges and solutions in bridging NLP and SQL databases.

Academic Research:
Researchers have written many papers discussing how to make these systems better and exploring new ways to combine language and databases. These papers help us understand the deeper technical problems and give us ideas on how to improve our chatbot.

Reviewing these previous projects shows us how the idea has grown over time and what the current technology can do. It also points out that there are still new things to be discovered and improved. Our project aims to add to this field by making a chatbot that is easy to use, helpful, and smart.

III. Dataset

The dataset for an NLP SQL Chatbot should include pairs of natural language queries and their corresponding SQL statements, reflecting diverse domains such as finance, healthcare, and e-commerce. It needs to capture database schema details, including tables, columns, and relationships, allowing the chatbot to map natural language intents to appropriate database structures. The dataset should include both simple and complex queries, ranging from basic SELECT statements to queries with JOINs, subqueries, and aggregations. Additionally, it should encompass realistic language usage, including variations, colloquialisms, and different phrasings, to enable the model to handle diverse linguistic patterns and to develop NLP SQL Chatbots capable of translating natural language queries into accurate SQL statements for effective database interaction.

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 training data includes a variety of language styles, including formal**,** informal, technical, and everyday language, allowing the Gemini model to handle diverse linguistic expressions. This diversity is crucial for applications such as NLP SQL Chatbots, as it enables the model to interpret and translate natural language queries into SQL statements accurately, regardless of the phrasing or complexity of the input.

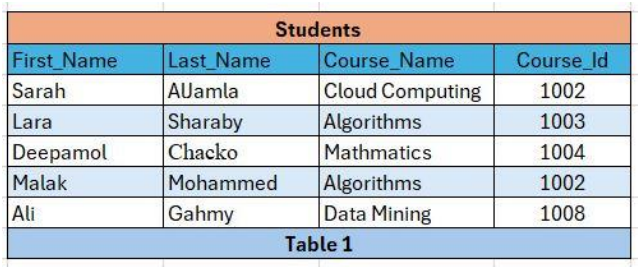
By training on such a diverse dataset, the Gemini model gains an extensive understanding of natural language, enabling it to generalize across various domains and contexts. This makes it an ideal choice for applications that require robust NLP capabilities, including NLP SQL Chatbots, where it can bridge the gap between natural language and database interaction seamlessly.

Overview

For our chatbot project, we use a simple database that acts like a small model of a Students data. This database includes information about students and the classes they enrolled in. It has two main tables: one for students and one for courses as shown in the images below.

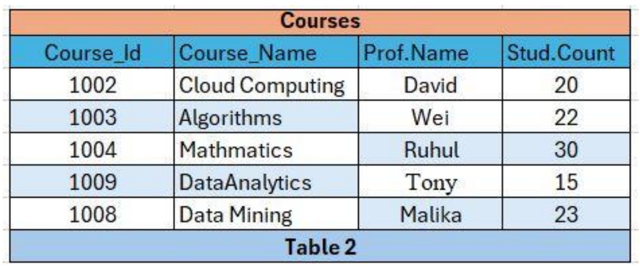
Student Table: This table keeps information about each student, like their first and last name, courses they are taking, etc. as shown in Table 1

Course Table: This table keeps information about each course, their ID's, name, Professor name, count of students who takes this course, etc, as shown in Table 2.



Students			
First_Name	Last_Name	Course_Name	Course_Id
Sarah	AUamla	Cloud Computing	1002
Lara	Sharaby	Algorithms	1003
Deepamol	Chacko	Mathmatics	1004
Malak	Mohammed	Algorithms	1002
Ali	Gahmy	Data Mining	1008

Table 1



Courses			
Course_Id	Course_Name	Prof.Name	Stud.Count
1002	Cloud Computing	David	20
1003	Algorithms	Wei	22
1004	Mathmatics	Ruhul	30
1009	DataAnalytics	Tony	15
1008	Data Mining	Malika	23

Table 2

How Tables are Connected?

The Course_id in the students table links each students to their course using the course id number. This helps us ask more complex questions about students and their courses.

How We Use the Tables in Our Project?

For Testing the Chatbot: We use these tables to check if our chatbot can understand and answer questions correctly. For example, we might ask, "Which course is taken by majority studnets?" or "What course does Lara Sharaby enrolled in?" The chatbot should be able to look at the tables and find the right answers.

To Make Things Better: Right now, our tables are small, but we can add more information/columns to them. This helps us see if our chatbot can handle bigger tasks and more data.

IV. Methodology

Implementation:

Our project utilizes a straightforward model to transform everyday language into database queries. As illustrated in our Figure1, it begins with a user inputting a question. This input is then processed by a large language model (LLM), which is trained to understand and interpret natural language. The model analyzes the question and generates an SQL query based on what it understands. Finally, the system executes this SQL query to retrieve the required data, and the results are displayed back to the user. This process makes it possible for anyone to interact with complex databases simply by asking questions in a way that feels natural and easy.

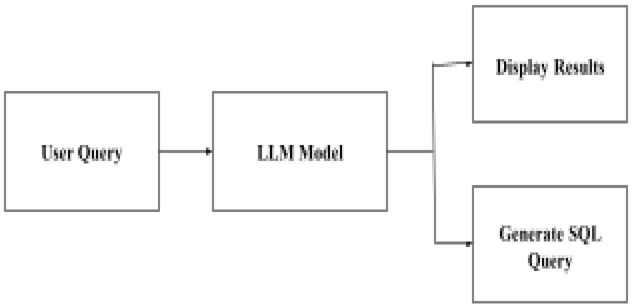


Fig1: Base Model

User Interface Design: The Streamlit application was designed to be intuitive, allowing users to input queries in natural language and view results directly on the web interface.

LangChain Integration with LLM Model: LangChain was configured to interface seamlessly with both the LLM and the SQLite database, ensuring that SQL queries are generated accurately and efficiently.
Optimization: Special attention can be given to the integration process to minimize latency and maximize the accuracy of query translations, leveraging LangChain's capabilities to handle natural language understanding and SQL generation.

Database Setup: The SQLite database was set up with multiple tables to represent complex data relationships and facilitate comprehensive query capabilities.

Our system architecture is designed to handle specific tasks efficiently as shown in the Figure 2.

- First, when a user types a query into the Streamlit interface, the system captures this input.
- Next, the input is sent to LangChain, which uses the large language model to understand the query and figure out what information the user is looking for.
- LangChain then turns this understanding into an SQL query, which is a set of instructions sent to the SQLite database.
- The database runs this SQL query to find the right data and sends it back to the chatbot.
- Finally, our Streamlit interface takes this data and displays it to the user in an easy-to-understand format.

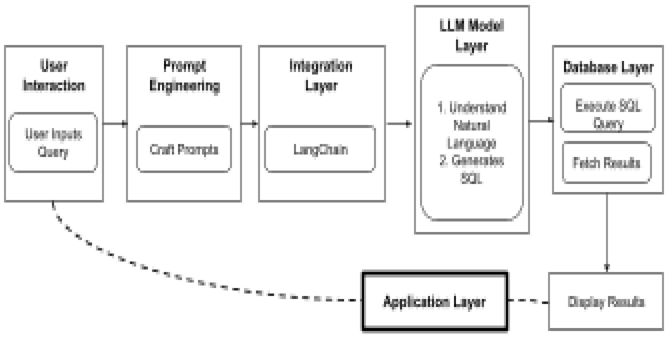


Fig 2: Model Architecture

Preprocessing
We created a 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.

V. Technologies Used

In this project, we use four main technologies to build our chatbot: LangChain, Streamlit, an SQLite database and Prompts. LangChain is a special tool that helps our chatbot understand and use large language models. These models are like very advanced algorithms that help the chatbot understand what people are asking in their natural language, like English. Once LangChain figures out what the user is asking, it turns this understanding into SQL queries. SQL queries are commands that get information from databases.

We chose Streamlit to create the part of our chatbot that people interact with. Streamlit is an open-source Python framework designed to simplify the development and deployment of interactive web applications, particularly for data science and machine learning projects. It offers a straightforward way to convert Python scripts into dynamic web apps with minimal code, making it ideal for creating user interfaces for machine learning models. It lets us build a user-friendly interface quickly and easily. This interface is where users can type their questions and see the answers. Streamlit works really well with our backend processes, which are handled by LangChain and the SQLite database.

SQLite is a lightweight, self-contained, and serverless relational database management system (RDBMS) widely used in various applications. It stores data in a single file, making it ideal for small-scale projects and applications that require a simple, embedded database solution. The SQLite database is where all the data we want to access is stored. It's set up to be simple and efficient, making it perfect for our needs in this project. We set up the database with multiple tables that represent different kinds of information an organization might have, like details about employees and departments. This setup allows us to simulate real-world data relationships and operations.

Our chatbot uses sophisticated technology to make it easy for anyone to retrieve information from a database by simply asking questions in natural language. This setup not only simplifies the process but also makes it quicker and more intuitive for users, whether they have technical skills or not. For this purpose, we guide the model using various prompts.

Prompt: A "prompt" in the context of NLP and machine learning, especially with models like GPT (Generative Pre-trained Transformer), is an input text sequence provided to the model to elicit a specific response or action. The prompt acts as an instruction or trigger that guides the model on what to perform next. The perfection of output generated by the model depends on how we phrase the questions to the model or how we handle the prompt engineering part. Below is an example of some prompts we plan to use.

ExamplePrompt=["""You are an expert in converting English questions to SQL query! The SQL database has the name Students and has the following columns - First_Name, Last_Name, Course_Name, Course_Id \n\nFor example,\nExample 1 - How many entries of records are present?, the SQL command will be something like this SELECT COUNT(*) FROM Students;\nExample 2 - Tell me all the students studying in Data Science class?, the SQL command will be something like this SELECT * FROM Students where Course_Name="Data Science"; also the sql code should not have ``` in beginning or end and sql word in output"""]

Prompt Engineering: A crucial part of making sure our chatbot works well is prompt engineering. It is a detailed process of crafting effective prompts that guide the LLM to generate accurate SQL queries from natural language inputs. This involves creating effective prompts or instructions that help the language model understand exactly what the user means. We carefully design these prompts to guide the model in generating accurate SQL queries. We continuously refine these prompts based on user feedback and the initial results we get during testing.

A well-crafted prompt for an NLP model converting text to SQL should include several key elements to guide the model effectively. The prompt should start by clearly describing the task, such as retrieving specific information from a database. It should mention the entities and attributes involved, such as the table and columns to be queried, and include any relevant conditions or filters. Additionally, it should specify aggregations or sorting requirements, if applicable. To provide further clarity, the prompt can include examples to illustrate the expected format and guide the user and model. Finally, mechanisms for clarifications or fallback responses should be integrated to handle ambiguous queries or misinterpretations, ensuring the model understands and completes the desired task accurately. For example, if you have an NLP model trained on a dataset of English text to SQL conversions, and you provide it with a prompt such as "Show me all orders from last year," the model recognizes this as a request for data retrieval, mapping it to a **SELECT** SQL statement and incorporating the "from last year" part as a WHERE clause to filter records by date. Through its training, the model has learned the patterns of how such requests translate into SQL queries and can generate a response accordingly.

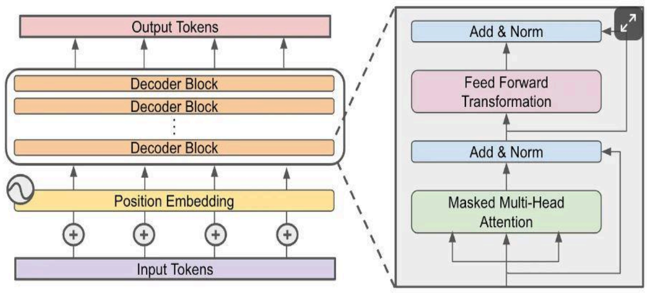
VI. LLM Gemini Pro Model

The Gemini model builds on the Transformer architecture, leveraging its powerful processing capabilities for NLP tasks. Pretrained on a vast corpus of text data, the Gemini model has learned language patterns, relationships, and linguistic nuances, providing it with a broad understanding of natural language. When an input query is provided, it first tokenizes it into smaller units, which are then converted into numerical embeddings for processing by the Transformer's self-attention mechanism. The model's NLP capabilities allow it to detect the intent behind the natural language query and map it to relevant database elements, facilitating the generation of accurate SQL queries. The model's architecture enables it to handle different types of SQL statements, including SELECT statements, JOINs, subqueries, and aggregations, making it ideal for NLP SQL Chatbots to interact seamlessly with databases. The Gemini model can also be fine-tuned on specific NLP tasks, like NLP SQL conversion, enhancing its performance by refining its understanding of specific domains and tasks.

The first generation of Gemini, known as "Gemini 1," consists of three models that share the same software architecture. These models are decoder-only transformers, modified to allow for efficient training and inference on TPUs. They offer a context length of 32,768 tokens and feature multi-query attention. Similar to many generative LLMs, Gemini is based on a decoder-only transformer architecture. Although the technical report does not explicitly detail the model's architecture, it provides enough information to roughly infer some relevant details.

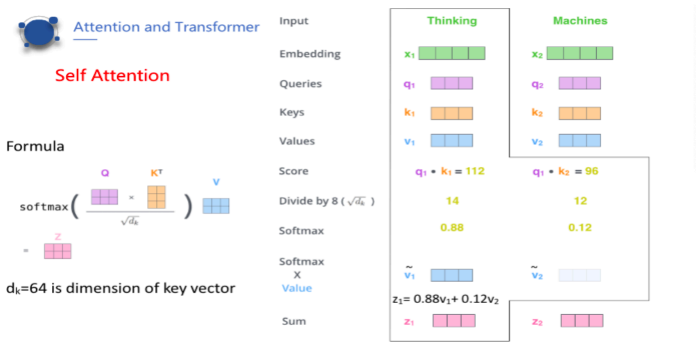
Transformers and Self-attention

The Transformer architecture, introduced in 2017, revolutionized NLP by providing a highly efficient model for processing and generating text. At its core is the self-attention mechanism, which enables the model to focus on different parts of the input text, regardless of their position in the sequence. This mechanism works by representing each word as a



The decoder-only transformer architecture

triplet: Query (Q), Key (K), and Value (V), which are derived from learned linear transformations of word embeddings. The model then computes a score for each Q-K pair, typically using a scaled dot product to measure similarity between the query and key embeddings. This score is normalized via a softmax function to produce a probability distribution that determines how much each value contributes to the output. These normalized scores are then used to create a weighted sum of the values, resulting in a new representation of the input sequence. The Transformer also incorporates multi-head attention, which allows it to focus on different parts of the input simultaneously, with each head having its own set of Q, K, and V transformations. Additionally, each layer of the Transformer includes a feed-forward neural network that applies non-linear transformations, adding depth to the model. Layer normalization and residual connections stabilize the training process, ensuring gradients flow smoothly and allowing inputs to be added directly to their outputs, making it easier for the model to learn effectively.



Langchain

LangChain is a Python framework designed to facilitate the creation and orchestration of applications that use large language models (LLMs). It provides tools for building modular chains of processing steps, enabling workflows that handle complex tasks, such as taking input, processing it through an LLM, and performing actions based on the output. LangChain offers flexible prompting mechanisms, memory capabilities for storing and retrieving information between tasks, and an agent framework that enables dynamic decision-making, delegating tasks to appropriate models or components. The framework also integrates seamlessly with other technologies, including database drivers, ORMs (Object-Relational Mappers), and APIs, streamlining workflows by connecting to external systems and services. Additionally, LangChain provides mechanisms for handling and displaying output, making it easier to present information to users in a meaningful way.

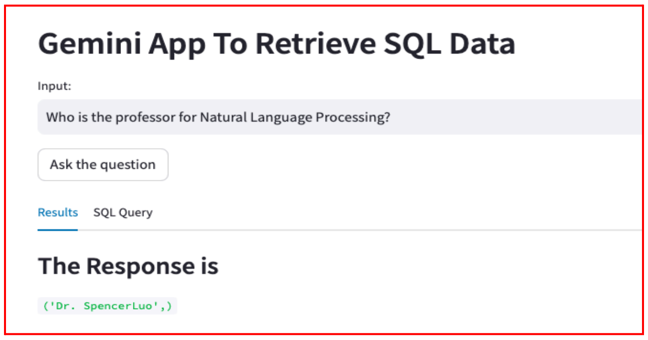
LangChain and Gemini Model

The Gemini model, developed by Google, is a standalone NLP model designed to handle various language-related tasks. Although not inherently tied to LangChain, LangChain can facilitate the integration of the Gemini model into an NLP SQL Chatbot. This Python framework enables the construction of modular chains of processing steps, including taking natural language input, processing it with the Gemini model, and executing the generated SQL query against a database. LangChain provides flexible prompting mechanisms that help the Gemini model translate natural language queries into SQL statements. Memory management features allow the chatbot to store and retrieve information between queries, maintaining context and enhancing accuracy. LangChain's agent framework enables dynamic decision-making and task delegation to appropriate models or components, including database drivers or ORMs for direct execution of SQL queries. This streamlines the workflow, making it easier to retrieve and present results in a user-friendly format. LangChain also supports fine-tuning workflows, enhancing the Gemini model's proficiency at specific NLP tasks, including NLP SQL conversion, ensuring seamless functionality across the chatbot's architecture.

VII. Demo 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 and text response as well.

Text Response



Gemini App To Retrieve SQL Data

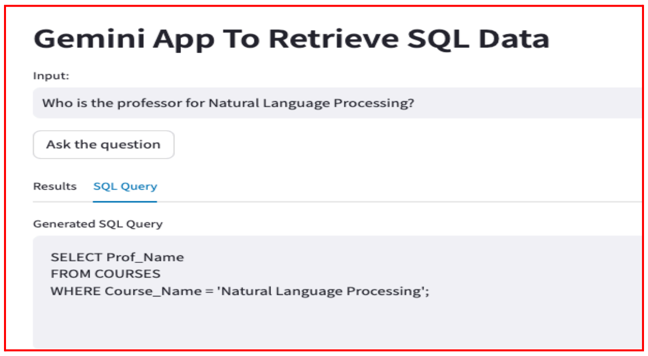
Input:

Who is the professor for Natural Language Processing?

The Response is

```
('Dr. SpencerLuo',)
```

SQL Query Response



Gemini App To Retrieve SQL Data

Input:

Who is the professor for Natural Language Processing?

Generated SQL Query

```
SELECT Prof_Name
FROM COURSES
WHERE Course_Name = 'Natural Language Processing';
```

VIII. Limitations

Creating an NLP-to-SQL chatbot project involves several challenges and limitations that must be considered. Natural language can be ambiguous, leading to incorrect SQL queries if the chatbot misinterprets the user's request. The chatbot needs a thorough understanding of the database schema, including table relationships, column types, and constraints, to avoid generating syntactically correct but logically flawed queries. Additionally, SQL queries can become highly complex, involving joins, subqueries, aggregations, and conditional logic, which can be challenging for the model to translate from natural language without extensive training. The project's effectiveness heavily depends on the diversity and depth of training data, and security risks, such as SQL injection, must be managed with robust mechanisms. The balance between generalization and specialization, user expectations, language variations, and the need for maintenance and updates are all challenges that must be addressed. Despite these limitations, NLP-to-SQL chatbot projects offer substantial value with careful design, comprehensive training, and robust error handling to ensure accurate, secure, and user-friendly operation.

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.

IX. Future work

The future of NLP2SQL involves several key developments. One area of focus is improving the accuracy of natural language to SQL translation, particularly for complex queries involving joins, subqueries, and aggregations. Another key area is domain-specific fine-tuning, which will enhance the model's understanding of specific industries like healthcare, finance, and e-commerce, making it more effective in these contexts. The integration of dynamic database schema mapping and metadata embedding can further broaden the range of databases these systems can handle. Planning to add more complex query handling and support for additional databases.

ChatBot with memory

is a tool that helps language models to remember things from one conversation to the next and use different resources to get information. This is really helpful for projects where you need the model to keep track of what was said before or to keep using information over several talks. It's like giving the language model a notepad to note down notes and a set of books it can refer to, which makes it smarter and more helpful across different interactions.

X. Conclusions

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

XI. References.

[1] Dr. Spencer W. Luo. Spring 2024 Natural Language Processing (CISC-6210) Lecture Slides.

[1] **"**Building Chatbots and Conversational Agents with SQL and AI". https://medium.com/@amb39305/building-chatbots-and-conversational-agents-with-sql-and-ai-4ca9204907d7

[2] F. Borges, G. Balikas, M. Brette, G. Kempf, A. Srikantan, M. Landos, D. Brazouskaya, and Q. Shi, "Query understanding for natural language enterprise search," 2020.

[3] M. Joseph, H. Raj, A. Yadav, and A. Sharma, "Askyourdb: An endto-end system for querying and visualizing relational databases using natural language," 2022.

[4] P. Wang, T. Shi, and C. K. Reddy, "Text-to-sql generation for question answering on electronic medical records," in WWW 2020, ser. WWW '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 350–361.