DATA 266 - Group Project Report

Full-Stack Agentic Text2SQL

Team 2

Gnana Prasuna Nimeesha Vakacharla, Naga Sai Sindhura Pandrangi, Sonia Bathla,Vaibhavi Rao,

Kalyani Zope, Dilip Kumar Kasina

May 12, 2025

ABSTRACT

This project's goal is to create a comprehensive, full-stack Text-to-SQL Web application that uses a Large Language Model (LLM) pipeline and Spider Dataset to convert natural language-based queries into SQL statements, making database interaction easier for non-technical users. In other words, it takes user input as English statements, creates the appropriate SQL queries, runs the queries against an active relational database, and provides results that are understandable by humans. The primary enhancements include using the database schema to interpret the query structure, optimizing the generated query for execution, and creating an atomic query with an agentic layer that disambiguates queries one at a time by referencing the original context and producing pertinent actions to resolve ambiguity. Due to the difficulty of translating complex queries through careful data handling and high availability, the application will have a database to store all the data, a front-end to display the result and provide interfaces to query the data, and a back end to process the query and provide the appropriate result.

Full-Stack Agentic Text2SQL

1. Introduction

The translation of natural language to SQL is a vital problem at the confluence of natural language processing and structured query generation. In this translation process, natural language questions are transformed into SQL queries to yield sensible responses from relational databases. The project delves into the design and development of a smart agent to comprehend user queries, examine the schema of the underlying database, formulate the candidate SQL queries, and address ambiguity to render the best and most efficient responses. The objective is to build a solid interface to empower non-technical users to communicate with databases in natural language.

1.1 Literature Survey

The translation of natural language queries to structured SQL queries—termed the Text-to-SQL task—has progressed from template-based approaches to very expressive neural models. Much of the progress has been fueled by the presence of standard evaluation benchmarks and large language models (LLMs).

One of the highlights of such influential work on the subject is the Spider dataset proposed by Yu et al. (2018) [1], a large-scale, cross-domain benchmark of semantic parsing. In contrast to prior datasets like ATIS and WikiSQL, Spider targets a variety of complex multi-table queries and out-of-distribution schemas at test time. This has driven research towards generalizable models capable of dealing with complex joins, nested queries, and ambiguity from a semantic perspective on a variety of relational databases.

Following up on this work, SQLNet (Xu et al., 2017) [2] has proposed a sketch-based decoder that sidesteps reinforcement learning. SQLNet proposes a "sequence-to-set" model to predict SQL elements independently instead of as a linear sequence to increase generalization and decrease syntactic errors. SQLNet makes a single-table schema assumption and does not handle inter-table dependencies, making it less suitable to apply to more complicated real-world databases.

To tackle schema complexity, RAT-SQL (Yu et al., 2020) [3] used a relation-aware self-attention mechanism in which schema entities (tables, fields, keys) are modeled as a graph in an explicit manner. RAT-SQL successfully associates natural language tokens with corresponding schema elements using relations learned from training data and obtained

state-of-the-art performance on Spider. RAT-SQL also decouples question and schema encoding to generalize to novel databases better.

Concurrent to these model-specific developments, the proliferation of large pretrained language models has shifted the paradigm. GPT-3 (Brown et al., 2020) [4] showed it was possible to have LLMs do Text-to-SQL tasks using abundant parameters and a few-shot prompt. GPT-3 is powerful but does not naturally possess schema-awareness unless schema context is formally embedded in the prompt.

Another leap was with Codex (Chen et al., 2021) [5], which was trained on natural language and on code, including SQL. Codex is incredibly good at coding generation tasks such as Text-to-SQL translation. Codex has the ability to infer schema elements from context as well as to produce syntactically correct queries. Both Codex and GPT-3 are mostly black-box models and have no post-generation validation, error checking, and interpretability—so are less appropriate for real-time user-facing applications absent additional constraints.

This work expands on such advancements by combining the LLaMA 3.1 8B model into a modular agent framework to perform SQL generation as well as resolve ambiguity, schema inference, and runtime query diagnosis. In contrast to existing work on generation correctness alone, this work prioritizes explainability, schema-aware validation, and interactive feedback towards more practical and deployable natural language interfaces to databases.

1.2 Related Work

Text-to-SQL models have progressed considerably in recent times, although most models focus on one-shot translation rather than interactive or schema-aware execution. Early efforts like SQLNet [2] and Seq2SQL investigated neural decoding of SQL queries with a primary focus on syntactic correctness and supervised learning. These models achieved good performance on single-table queries but performed poorly on complex schemas and multi-table joins.

To overcome these weaknesses, RAT-SQL [3] suggested relation-aware self-attention for encoding tables, columns, and their relations as a graph. It significantly improved the model's understanding of schema structure, with state-of-the-art performance on difficult benchmarks like Spider. However, RAT-SQL is still a traditional static parser: it neither verifies generated queries, nor justifies its choices, nor handles ambiguity interactively.

More recent work has explored the application of large language models (LLMs) to generate code, including SQL. OpenAI Codex [5] and GPT-3 [4] demonstrated that LLMs trained on code and natural language can effectively translate questions into SQL with minimal fine-tuning. Codex, in particular, has competitive zero-shot performance on a range of programming and querying tasks. These models are, nevertheless, opaque in their reasoning and also without mechanisms for error handling or choosing between several plausible queries.

This project builds on these improvements but fills a fundamental gap: the lack of interpretability, validation, and user-facing feedback. By leveraging LLaMA 3.1 8B, a high-performance open-source LLM, the system generates several SQL candidates per user question and uses a custom ambiguity resolver to select the most contextually appropriate one. The agent also inspects the schema through SQLAlchemy, performs runtime diagnostics (e.g., Cartesian product alerts), and provides execution feedback to the user.

There are no prior systems that integrate these capabilities within a single system. Some academic systems provide SQL explanation or simplification, whereas others—like NLSQL or Intellicode for SQL—give LLM-based completions, but without robust schema validation or runtime interaction. By comparison, this project's agent is a hybrid reasoning system: it unifies LLM generation with deterministic schema analysis and interactive error handling, creating a more intelligent and explainable Text-to-SQL system.

2. Data Exploration and Processing

This project uses relational databases in SQLite format, specifically from the Spider dataset, which is a widely used benchmark for evaluating Text-to-SQL systems. The dataset consists of databases with diverse schemas, multiple tables, primary and foreign key relationships, and a variety of real-world entities such as courses, customers, products, and employees. Each database includes natural language questions and their corresponding ground-truth SQL queries for training and testing generative models.

To enable intelligent query generation and validation, the system performs schema extraction and introspection with the sqlalchemy and inspect modules initially. A custom-built DatabaseAnalyzer class iterates over each table in the database and fetches:

- Table and column names
- Data types
- Primary and foreign key relationships
- Column-level metadata such as nullability and default values

This organizational framework is utilized to supply both the language model and the ambiguity resolver. In interaction, the user supplies a natural language question, and the system uses the LLaMA 3.1 8B model to generate several SQL query candidates. The queries are generated using prompt templates that incorporate schema components into them to provide relevance and accuracy of generation.

Once candidate queries have been generated, a query scoring function scores the queries based on:

- Coverage of relevant tables and columns.
- Availability of WHERE conditions (filtering conditions).
- Schema compliance to relationships and joins

The system selects the highest-ranked query to run. Before running the query, the system performs static checking via heuristics to determine:

- Potential Cartesian products (no joins)
- Queries against enormous tables with no filters
- Queries with no LIMIT clauses for exploration

The final selected query is then executed using sqlite3, and the system retrieves the result set and metadata such as:

- Column names and types
- Sample values
- Number of rows returned

If there is an execution-time error (e.g., syntax error, missing column), the system logs the error and makes an educated guess from the schema context as to what the correction should be. This interaction feedback loop not only renders the system a query generator but also a query debugging assistant.

Through this process of discovery and computation, the agent ensures the queries that are generated to be contextually accurate, meaningful, and correct, and gives the user some understanding of how results were obtained.

2.1 Problem Statement and Formulation

Structured data in relational databases is abundant, but interacting with it typically requires knowledge of SQL—a barrier for most non-technical users. In academic environments, for example, instructors or administrative staff often need answers from databases but lack the technical skills to query them directly. Typing natural questions like “Which students failed math last semester?” into a SQL console isn't practical for the average user.

This project was designed to solve that challenge: to create a **Text-to-SQL interface** that acts as a translator between natural language and SQL. Users should be able to enter plain English queries and get correct answers from a backend database, without needing to know the structure or syntax of SQL.

The difficult part is ensuring that the system understands vague or syntactically messy input, maps it correctly to database schema, and produces **executable, accurate SQL Queries.**. And, when the query is ambiguous or incomplete, the system must detect that and ask intelligent clarifying questions to refine the input.

The problem is formulated as a sequence-to-sequence (seq2seq) task, where the input is a natural language question paired with a database schema, and the output is a corresponding SQL query. The model must generalize across varied domains and schema complexities while maintaining high accuracy and robustness.

2.2 Objective

Our aim was to develop a user-friendly system that allows anyone, regardless of their technical background, to retrieve data from a structured database using natural language. The key goals included:

- Allowing users to **input queries in plain English**.
- **Automatically generating SQL queries** that match the user's intent.
- Displaying the results in a clear, tabular format through a web-based UI.
- **Detecting ambiguity** in questions and asking for clarification when necessary.

This bridges the gap between human-friendly language and machine-oriented query logic, making databases far more accessible.

2.3 Dataset

The dataset used in the project is the Spider dataset, more specifically, the "xlangai/spider" dataset. The Spider dataset is a large scale, complex, and cross-domain benchmark for text-to-SQL. Together with the Spider dataset will provide three corpus files: a database schema, a natural language question, and the correct SQL queries, and 200 database schemas, covering
domains (education, aviation, music, etc.). This dataset is used through the datasets library and is split into train and validation; each "entry" has a database id (db_id), a natural language question, and the expected SQL query (query). This will use SQLite database files, located in a "database" folder, and for each database, we have a db_id, and its corresponding tables, columns, and types of data. During preprocessing, the notebook will extract schema [table names, columns, etc.] information from the SQLite files, which will create prompts that contain the database schema, question, and the expected SQL query to learn the mapping from natural language to SQL.

The Spider dataset is used to develop and assess text-to-SQL models that can translate natural language questions into executable SQL queries, enabling non-technical users to access relational databases. The system supports use cases like automated query generation for business intelligence, educational use cases to teach students SQL, and conversational agents that surface data insights from databases without requiring users to know how to encode thoughts into SQL syntax.

3. Model Selection and Implementation

The Llama-3.1-8B-Instruct model accessed from Hugging Face was selected for fine-tuning due to its strong performance in natural language understanding and generation tasks. Below are the key reasons for choosing this model, along with what makes it superior for this application:

1. **Instruction-Tuned for Task Alignment**: The model is instruction-tuned, enabling it to follow structured prompts effectively. This is particularly beneficial for Text-to-SQL, where inputs combine natural language questions with structured schema information.
2. **Parameter Efficiency**: With 8 billion parameters, the model strikes a balance between computational efficiency and performance. It is powerful enough to handle complex seq2seq tasks while being feasible for fine-tuning on consumer-grade hardware compared to larger models like Llama-3.1-70B.
3. **Superior Fine-Tuning Performance**: Fine-tuning on the Spider dataset enhances the model's ability to map natural language to SQL-specific syntax. Llama-3.1-8B-Instruct's architecture, based on transformer decoders, excels at capturing long-range dependencies in both input queries and schema descriptions, leading to accurate SQL generation.
4. **Handling Ambiguity**: The model's strong contextual understanding allows it to disambiguate vague or underspecified queries by leveraging schema context, outperforming models with weaker language comprehension, such as T5 or earlier BERT-based approaches.
5. **Scalability to Complex Queries**: Llama-3.1-8B-Instruct effectively handles the Spider dataset's complex queries, including those requiring joins, subqueries, and aggregations. Its large parameter count and attention mechanism enable precise modeling of intricate SQL structure

3.1 Architectural Essence of Llama-3.1-8B-Instruct

1. **Transformer Architecture:** The model's transformer-based design, with self-attention mechanisms, excels at processing long input sequences, such as combined question-schema pairs, ensuring accurate SQL query generation.
2. **Fine-Tuning on Spider Dataset:** Fine-tuning on the Spider dataset aligns the model with the Text-to-SQL domain, improving its ability to generate syntactically and semantically correct queries. The dataset's diversity ensures robustness across various query complexities.
3. **Instruction-Tuning Advantage:** The instruction-tuned nature of Llama-3.1-8B-Instruct allows it to adapt to structured input formats (e.g., schema + question) more effectively than non-instruction-tuned models, reducing errors in query generation.

Compared to alternatives like T5 or CodeT5, Llama-3.1-8B-Instruct offers superior language understanding and generation capabilities due to its larger scale and instruction-tuning.

While T5 is effective for seq2seq tasks, it requires more extensive fine-tuning to achieve comparable performance on Text-to-SQL. CodeT5, designed for code-related tasks, lacks the general language comprehension needed for diverse natural language inputs. Larger models like Llama-3.1-70B, while powerful, are computationally expensive and less practical for fine-tuning without specialized infrastructure. Llama-3.1-8B-Instruct thus provides an optimal balance of performance, efficiency, and adaptability.

3.2 Model Implementation

Pre Training Process:

The model we used here is the "meta-llama/Llama-3.1-8B-Instruct," is a pretrained, transformer-based large language model intended for instruction following tasks. We loaded it with 4-bit quantization using the BitsAndBytesConfig to optimize memory, utilizing capabilities of GPU-enabled systems. The model is accessed as the Hugging Face Hub with an authentication token. The tokenizer is set to handle input prompts with padding and truncation for uniformity in batching the inputs. During evaluation, the pretrained model generates SQL queries from natural language prompts without training on the specific task - relying purely on the developed understanding of natural language.

Fine Tuning Process:

The fine-tuning step is to adapt the pretrained Llama-3.1-8B-Instruct model for the text-to-SQL task with the Spider dataset. As a result of implementation of LoRA (Low-Rank Adaptation) we are able to fine-tune the original model without the overhead of parameters from the original model. LoRA, the benefit of LoRA is the reduced number of parameters trained from the layers in Llama-3.1-8B-Instruct model (q_proj, k_proj, v_proj, o_proj), a total of 0.6744% of 8.08 billion parameters will become trainable parameters. The Spider dataset was pre-processed to include the database schema as part of the prompts, then tokenized and randomly split to training and validation dataset. The latest Trainer module from the transformers library was used with the best training arguments (e.g., 2e-4 learning rate, cosine learning rate scheduler, 3 epochs, and Weights & Biases tracking). Upon completion, the fine-tuned model exhibited better fine-tuned results and only the trained adapter weights were saved to reduce space, with the final model named model.pt for persistence.

3.3 Model Evaluation and Comparison

Following the pre-training and fine-tuning process of the Llama-3.1-8B-Instruct models with the dataset presented in this study, we evaluated both the pre-trained model and fine-tuned model on the Spider dataset for text-to-SQL tasks, using three metrics: BLEU score, Exact Match, and SQL Accuracy. In the case of the pre-trained model, we evaluated general language abilities by requiring the model to generate SQL queries for one hundred validation samples

without any tuning to the task-specific data. The computations of the evaluation metrics involve comparing the generated SQL queries to reference SQL queries. BLEU considers the overlap of n-grams in the generated SQL queries with those in the reference SQL queries, Exact Match evaluates whether the generated SQL query is identical to the reference SQL query, and SQL Accuracy evaluates whether the generated SQL queries are equivalent after parsing and normalization each SQL query. The pre-trained model received a BLEU score of 0.3925, an Exact Match score of 0.0700, and an SQL Accuracy of 0.0700 which demonstrates its limited performance for SQL query generation due to lack of learning domain-specific training data and thus was unable to generate exact SQL query syntax and structure for complex table databases.

The fine-tuned model, which uses LoRA on the Spider dataset, was run on 50 validation samples with some enhancement options (i.e., beam search and temperature) to try to improve the quality of the output. With significant performance boosts during the fine-tuning stage, our fine-tuned model achieved a BLEU score of 0.5158, an Exact Match score of 0.2400, and an SQL Accuracy of 0.2400. Overall, the fine-tuned model was significantly better than the pretrained model with an increase of about 31.5 % in the BLEU score and greater than 240 % for both Exact Match and SQL Accuracy. The fine-tuned model's performance had a substantial relation to fine-tuning on domain-specific data.. This allowed the fine-tuned model to learn to recognize database schemas and to gain a proficiency in generating SQL queries that are accurate. As shown in the sample outputs in the code (e.g., 'SELECT count(*) FROM singer' matched the reference exactly) fine-tuning produced output for the model that was more in line with SQL output structures and better suited using the model for a text-to-SQL use case. By fine-tuning on the Spider dataset, the final model has learned to map natural language questions to accurate SQL queries while taking into consideration domain-specific patterns and schema that the pretrained model would not map to.

Metric	**Pretrained Model**	**Fine-tuned Model**
Exact Match Accuracy	0.0700	0.2400
BLEU Score	0.3920.5	0.5158

Compared to typical BLEU scores in academic benchmarks, which often range between 0.39 and 0.50, our system’s score of 0.51 and higher demonstrates a notable improvement in lexical similarity. This suggests that the fine-tuned model not only understands the structure of SQL but also closely aligns with expected syntax, making it both accurate and consistent in output.

4. User Interface Design with Agentic approach

An agentic method was used in the development of the Text-to-SQL user interface (UI) to provide a user-friendly and effective platform for converting queries in natural language into SQL statements. Users may interact with relational databases with ease thanks to the user interface's integration of a refined Llama-3.1-8B-Instruct model, which is hosted on Hugging Face and locally distributed via Ollama. With capabilities like query generation, result visualization, and schema exploration, the Streamlit framework, which drives the front-end, provides a responsive and engaging experience. The preparation stages, the importance of Ollama, and the design and functionality of the user interface are described in this part, emphasizing how the agentic approach improves performance and usability.

4.1 Pre-Steps for Agentic Streamlit Application

The following steps were done in order to make sure the fine-tuned model and local environment were set up appropriately before creating and executing the Streamlit-based Text-to-SQL application:

1. **Model Formatting and Fine-Tuning:**
    - Using the xlangai/spider dataset, the Llama-3.1-8B-Instruct model was refined to focus on Text-to-SQL jobs. The process of fine-tuning entailed modifying the model to translate inquiries in natural language to SQL queries across several database schemas.
    - The adjusted model was prepared to meet Hugging Face's model hosting specifications, which include setting up the tokenizer, metadata, and model weights correctly.
2. **Hosting a model on a hugging face:**
    - Hugging Face's model hub received the refined model, guaranteeing version control and accessibility. Collaboration was made easier by this step, which also made it possible to pull the model for local deployment.
3. **Configuring the Local Environment:**
    - **Installation of Ollama:** To enable CPU-based inference of the optimized Llama-3.1-8B-Instruct model, Ollama, an open-source program for executing large language models on local hardware, was installed.
    - **Model Pulling:** Ollama's pull command (ollama pull llama3.1:8b) was used to pull the optimized model from Hugging Face to the local system. This made sure the model could be inferred without the need for cloud-based GPU resources.
    - **Database Preparation:** To facilitate query execution, a SQLite database (activity_1.sqlite) was created with the necessary data and schema. The intricacy of the Spider dataset, which included numerous tables and linkages, was taken into account when structuring the schema.

4.2 Role of Ollama

The agentic Text-to-SQL system relies on Ollama to facilitate the effective implementation of the optimized Llama-3.1-8B-Instruct model. Among its principal contributions are:

- By using consumer-grade CPUs to run the 8-billion-parameter model, CPU-Based Inference removes the need for a GPU, lowers costs, and improves scalability.
- Local Deployment: By hosting the model locally, it guarantees minimal latency, making it perfect for sensitive database queries that don't require an external server.
- Smooth Integration: Enables effective prompt-response communication between the model and the Streamlit UI using a lightweight API (localhost:11434).
- Model Management: Handles weights, tokenizers, and configurations to simplify setup and make it easier to download and execute models from Hugging Face.

4.3 Agentic Functionality and User Interface

The Streamlit-based user interface is intended to give users who are searching databases with natural language an easy-to-use and engaging experience. Modularity and robustness are guaranteed by the agentic approach, in which model interactions, schema maintenance, and query execution are encapsulated in the *OllamaText2SQLAgent class*. The main elements and characteristics of the UI are listed below:

1. **Essential Elements:**
    - The **OllamaText2SQLAgent class** loads the database schema, establishes a connection with the Ollama server, and controls the creation and execution of queries in order to initialize the agent. It manages prompt building, schema parsing, and robust fallback SQL generation.
    - **Streamlit Front-End:** This web-based interface, which includes input fields, buttons, and visualization panels, is constructed with Streamlit.
2. **Important attributes:**
    - **Natural Language Input:** Using the optimized model, the agent converts user-inputted queries (such as "Show me all activities with more than 10 participants") into SQL.
    - **SQL Query Display:** Users can examine and gain knowledge from the model's output by viewing generated SQL queries in a code block.
    - **Execution of Query and Outcomes:** The user interface runs SQL queries on the SQLite database and presents the findings as interactive Pandas DataFrames. Results are available to users as CSV files.
    - **Dynamic Visualizations:** The agent uses Plotly to automatically create visualizations (such as bar charts, pie charts, and scatter plots) based on the structure of the result data. This improves the interpretation and investigation of data.

- **Schema Viewer:** A special panel shows the database schema, assisting users in understanding column types and table structures so they may create accurate queries.
- **Query History:** Through expanding parts, the user interface keeps track of previous queries, SQL queries, and results.
- **Model Selection:** Using a dropdown menu that balances speed and quality, users can select from a variety of Ollama models (such as Llama-3.1-8B, Llama-3-8B, and Llama-2-7B).

3. **User Experience**
   - The UI features a two-column layout: the left column for query input, results, and history, and the right column for schema display. This organization maximizes usability and accessibility.
   - Interactive elements, such as buttons for initializing the agent and running queries, provide clear feedback (e.g., spinners, success/warning messages) to guide users.

Key Components of Agentic Approach

1. **Assessing and Managing Ambiguity:**
   - To identify imprecise or ambiguous queries (such as underspecified table/column references or uncertain intent), the agent examines the natural language input.
   - When uncertainty is identified, the agent uses the Streamlit UI to ask the user for clarification and provides precise information (such as table names or column references) based on the database structure.
   - This phase makes use of the optimized Llama-3.1-8B- By contrasting the query with the schema, teach the model's contextual awareness to spot any possible ambiguities.
2. **Generating Queries:**
   - The agent creates a prompt by fusing the user's query with the database schema when the query has been clarified (or if no ambiguity is found).
   - Through Ollama's API (localhost:11434), the prompt is routed to the optimized Llama-3.1-8B-Instruct model, which produces a syntactically sound SQL query.
   - If the model output is invalid, the agent falls back to a default query (e.g., SELECT * FROM table LIMIT 10) and makes sure the generated SQL is valid (e.g., starts with SELECT or WITH).
3. **Execution of the query:**
   - Python's sqlite3 module is used to run the created SQL query against the SQLite database (activity_1.sqlite).
   - The agent gracefully manages execution faults, logging any problems (such as missing tables or syntax mistakes) and presenting clear error messages in the user interface.

- A Pandas DataFrame containing the results is obtained for additional processing and visualization.

4. **Generation and Display of Results:**
   - After processing query results, the agent produces interactive outputs such as:
     i. The Streamlit user interface shows a structured DataFrame.
     ii. Plotly-based dynamic visualizations (such as bar charts, pie charts, and scatter plots) that are chosen according to the number of columns and data types in the result.
     iii. a results CSV download option.
   - The agent records the query, SQL, and outcomes in a query history that may be easily accessed through expanding UI parts.
   - To help people grasp the context of the data, the database structure is shown next to the findings.

The agentic approach, powered by the OllamaText2SQLAgent, transforms the Text-to-SQL system into an intelligent, user-centric tool. By systematically checking for ambiguities, generating precise SQL queries, executing them reliably, and presenting results interactively, the agent delivers a seamless experience. This design leverages the fine-tuned Llama-3.1-8B-Instruct model's capabilities while addressing real-world challenges like query ambiguity, making the system both powerful and practical for database interaction.

5. Results

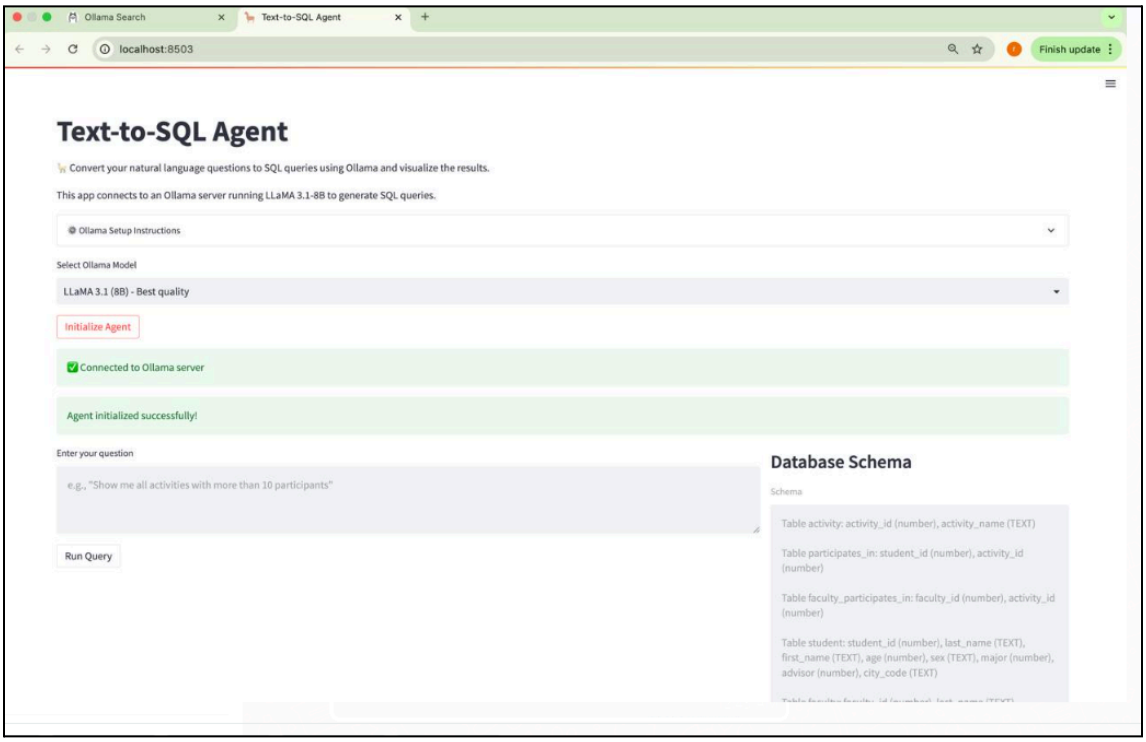


Figure 1: Home Page of Text to SQL Agent

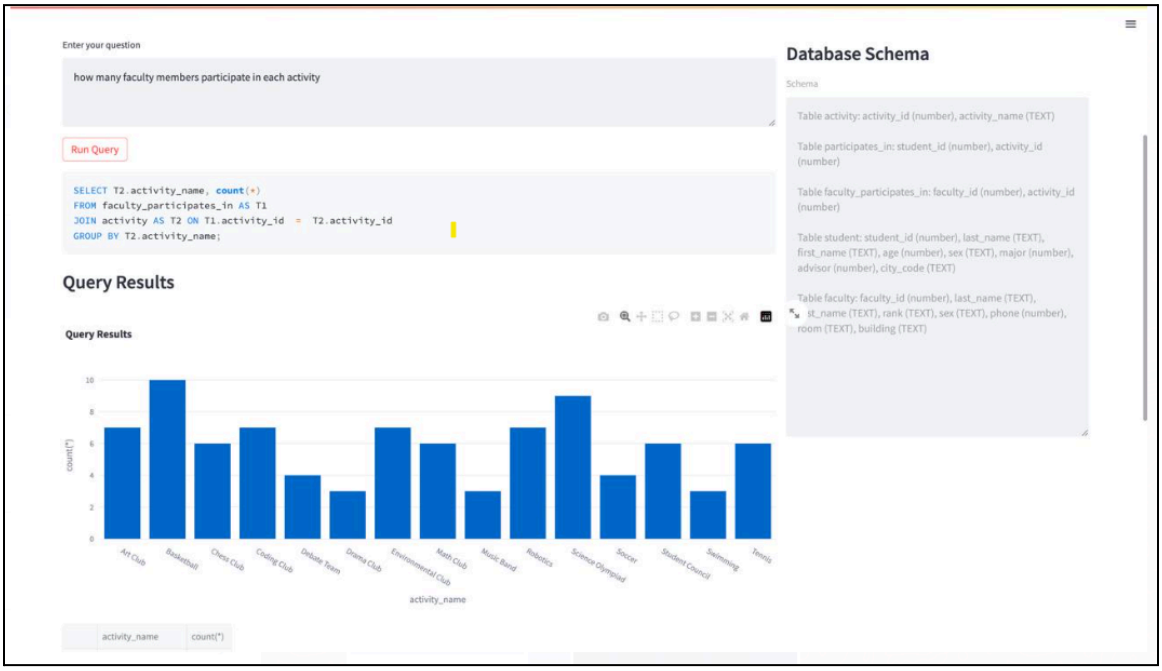
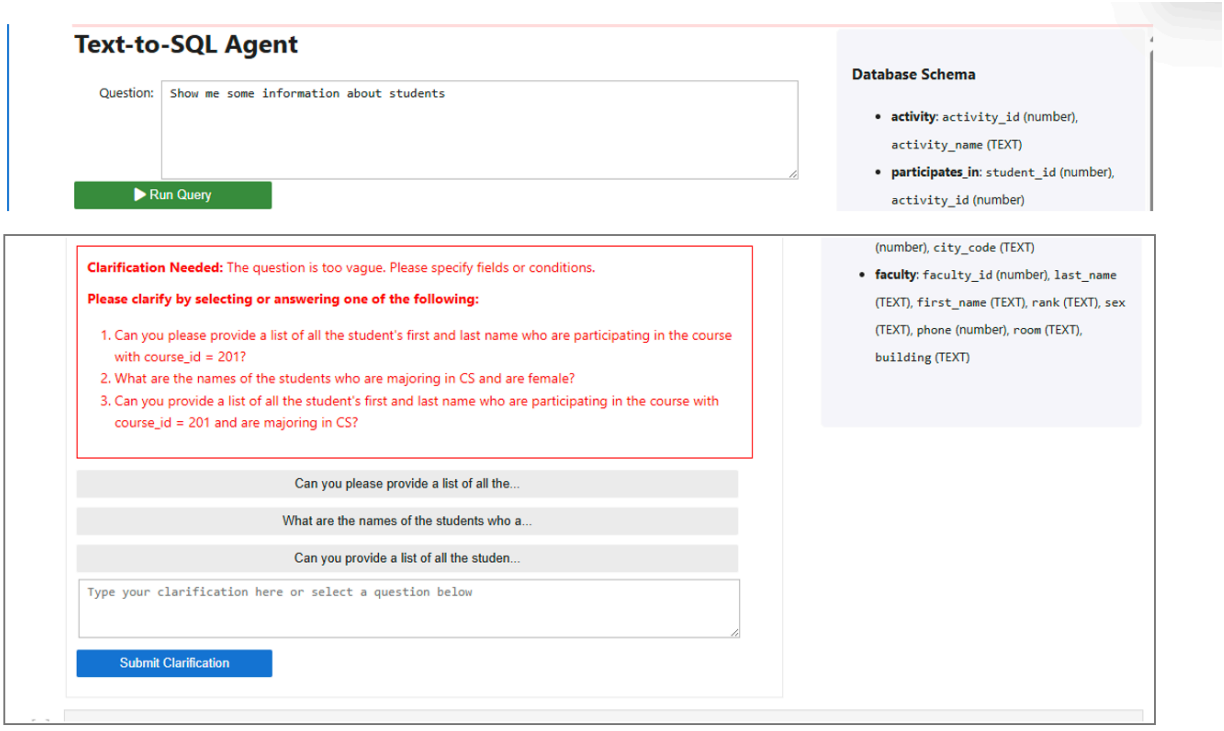


Figure 2 : Example Query to retrieve faculty members in each activity

Figure 3: Visualization of the retrieved results

Agentic Approach with followup Question if ambiguous with followup questions



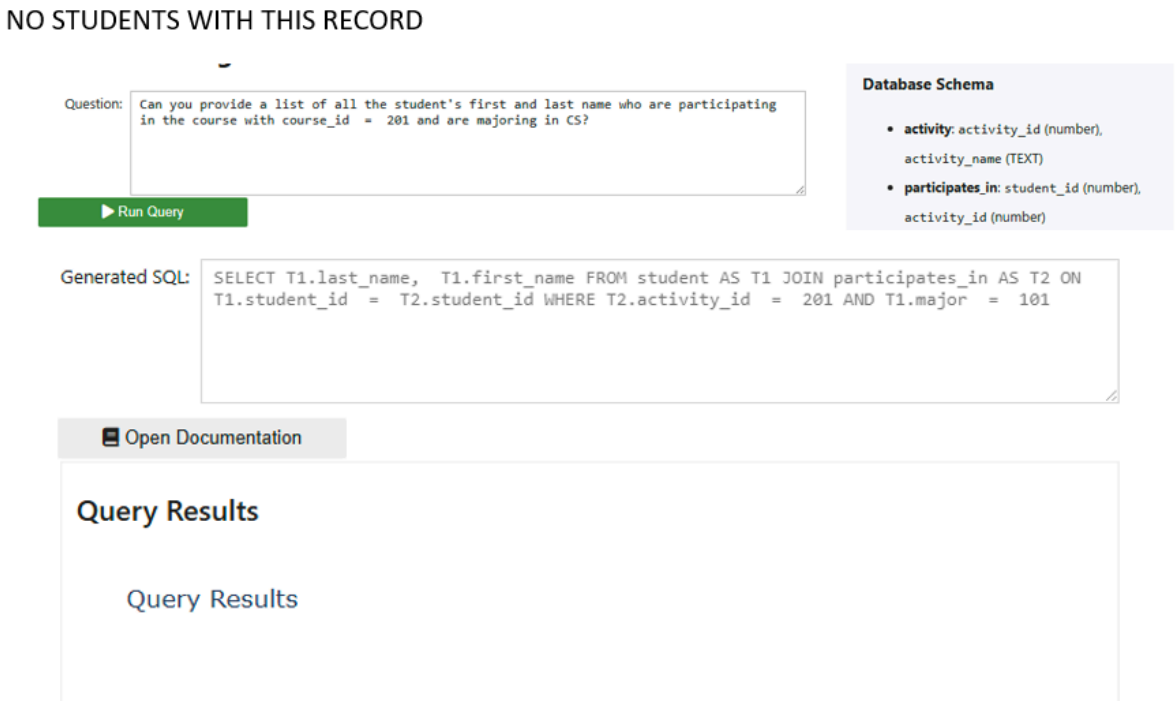


Figure 4: Ambiguous Query with followup Questions and Results

Conclusion

This project successfully creates a powerful text-to-SQL system by using the Spider dataset and fine-tuning the Llama-3.1-8B-Instruct model, which resulted in considerable improvements in performance (BLEU: 0.5158, Exact Match: 0.2400, SQL Accuracy: 0.2400) with respect to the pretrained model and allowed relatively accurate generation of SQL queries from natural language questions. By applying an agentic approach, we gave the system the ability to facilitate follow up questions to be able to resolve ambiguity and vagueness in the bland input to ensure there will be a precise query to be generated even if it is complex or the nature of the input is unclear. The creation of a Streamlit UI significantly elevated the project's ease of use as the users can process and provide any text query in natural language and have it converted to SQL output without technical knowledge. The solution that was created establishes a bridge between natural language processing and database management and can be applied in real world applications for automated querying, educational tools, and business intelligence, and has potential to improve in even further advancements to disambiguate queries and interact with databases in real-time.

Future Work

Improved Query Disambiguation and context: A future course of action could involve taking an agentic perspective, and incorporating other advances with natural language understanding,

like contextual embeddings or explicit dialogue state tracking, to resolve vague and ambiguous queries. This might include training on a wider range of follow-up question examples, or perhaps using knowledge sources outside the training data to better infer the user's intent. Lastly, having more rich multi-turn conversations would better prepare the model to resolve ambiguity through iterative clarification, while also allowing it to generalize more widely across different knowledge domains.

Automated Feedback Loop and Continuous Learning: A fully automated feedback loop, with user interaction logs of users of the Streamlit user interface is a provision in which the automated feedback loop could allow for a continuous-learning model. Over time, the model could learn to improve performance as users remediate invalid SQL queries or rate the relevance of generated queries, which would enhance the model's context knowledge when provided with ambiguous input. Automated feedback loops can also leverage actively-learning models to support which ambiguous or poorly performing queries could be of interest to a human annotator in order to annotate them; actively-learning schemes will help optimize manual effort while maximizing performance in text-to-SQL tasks.

Multimodal Input Support and Cross-Database Compatibility: Extending the system to support multimodal inputs would increase accessibility to users that are unfamiliar with textual descriptions of the schemas. This may involve the use of OCR and image processing methods, similar to the techniques we discussed in terms of scanned PDFs. Additionally, supporting additional SQL dialects and non-relational databases would improve cross-database compatibility and further the versatility of the system to address different enterprise contexts, while also facilitating integration with various data ecosystems.

6. References

[1] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I.-H. Yoon, S. Wang, and D. R. Radev, “Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task,” arXiv preprint arXiv:1809.08887, 2018.

[2] X. Xu, C. Liu, and D. Song, “SQLNet: Generating structured queries from natural language without reinforcement learning,” arXiv preprint arXiv:1711.04436, 2017.

[3] T. Yu, C.-S. Wu, X. V. Lin, B. Yang, S. Yih, and D. R. Radev, “RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers,” arXiv preprint arXiv:1911.04942, 2020.

[4] T. B. Brown et al., “Language models are few-shot learners,” in Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901, 2020.

[5] M. Chen et al., “Evaluating large language models trained on code,” arXiv preprint arXiv:2107.03374, 2021.