A MINI PROJECT REPORT

On

“Text-To-SQL-Converter”

Submitted in partial fulfillment of the requirements of the degree

BACHELOR OF ENGINEERING IN COMPUTER ENGINEERING

By

Sakshi Aher	**(101/123CP3130B)**
Garv Balgi	**(106/123CP3081A)**
Aayush Chalke	**(120/123CP3047A)**

Under the guidance of

Prof. Dhanashree Kane

Department of Computer Engineering

MGM’s College of Engineering and Technology,

Kamothe, Navi Mumbai- 410209

University of Mumbai (AY 2025-26)

CERTIFICATE

This is to certify that the Mini Project entitled **“Text-To-SQL-Converter”** is bonafide work of **Sakshi Aher (101/123CP3130B), Garv Balgi (106/123CP3081A),Aayush Chalke (120/123CP3047A)** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **“Bachelor of Engineering”** in **“Computer Engineering”.**

(Prof. Dhanashree Kane)

Guide

(Dr. Rajesh Kadu)	**(Dr. Geeta Lathkar)**
Head of Department	Director

Contents

ABSTRACT

The proliferation of data stored in relational databases has created a significant accessibility challenges for non-technical users who lack the expertise to write comple SQL queries.This project presents a Text-To-SQL converter,a system designed to empower users to interact with databases using natural language.By leveraging advanced natural language processing techniques and the power of large language model,our converter translates users queries posed in plain English into executable SQL statements.

The system is architected to first parse the user's intent, identifying key entities and relationships within the query. This is followed by a generative step where, with an understanding of the underlying database schema, a precise SQL query is constructed.

The converter is designed to handle a variety of query complexities, from simple data retrieval to more complex operations involving joins, aggregations, and filtering.

The primary objective of this project is to democratize data access, enabling a broader audience to harness the power of data without the steep learning curve of a formal query language. This work demonstrates a practical application of machine learning to enhance human-computer interaction and streamline data-driven decision-making.

ACKNOWLEDGMENTS

The satisfaction that accompanies that the successful completion of any task would be incomplete without the mention of people whose ceaseless co-operation made it possible, whose constant guidance and encouragement crown all effort with success.

We are greatly indebted to our **Director Geeta Lathkar** for all her encouragement, support and facilities provided in college that gave us enough enthusiasm, confidence and strength in getting this project to its present stage. Developing a project is not an easy task. Nothing of this project is possible without respected **H.O.D. Dr Rajesh Kadu** for encouraging us and giving this opportunity to widen our knowledge by this project. We give her special thanks from the depth of our heart.

We are also thanking you to our project guide **Prof. Dhanashree Kane**
for guiding us in our project.
We also thank our colleagues who have helped in successful completion of the project.

1. INTRODUCTION

1.1 Background

In the modern digital economy, data is a cornerstone of strategic decision-making, operational efficiency, and competitive advantage. Organizations collect vast amounts of information, which is typically stored in highly structured relational databases. The primary means of interacting with these databases is through **Structured Query Language (SQL)**, a powerful but specialized programming language.

1.2 Purpose

The purpose of this project is to eliminate the technical barrier between users and their data. A vast amount of valuable information is stored in relational databases, but accessing it requires proficiency in Structured Query Language (SQL), a skill many business analysts, managers, and other professionals lack. This project aims to develop an intelligent Text-to-SQL converter that acts as a translator, allowing users to ask questions in plain, natural English and receive the data they need. The ultimate goal is to empower non-technical users with self-service data access, thereby accelerating decision-making and fostering a more data-driven culture.

1.3.1 General Objective

The general objective of this project is to design and develop an intelligent system that translates natural language questions into executable SQL queries. By leveraging advanced Natural Language Processing (NLP) and machine learning models, the project aims to create a seamless and intuitive interface for non-technical users to interact with relational databases, thereby democratizing data access and eliminating the need for manual SQL coding for data retrieval.

1.4 Scope of the study

This study's scope is to develop a prototype system that translates simple English questions into executable SQL SELECT queries using NLP techniques. The system's functionality and evaluation will be limited to a single, predefined database schema to ensure a controlled testing environment.

2. LITERATURE SURVEY

2.1 Survey of Existing System/SRS

The manual method of data retrieval from relational databases forms the baseline existing system. In this traditional workflow, a user requiring specific data must formally request it from a database administrator (DBA) or a data analyst who is proficient in SQL. The analyst then writes, tests, and executes the query, finally exporting the results for the user. The primary drawback of this system is its inefficiency, leading to significant delays (high turnaround time), dependency on technical personnel, and a bottleneck that discourages data exploration.

In recent years, several automated systems have emerged to address this challenge, which can be broadly categorized:

1. **Commercial Business Intelligence (BI) Platforms:** Tools like **Tableau**, **Power BI**, and **ThoughtSpot** have integrated Natural Language Querying (NLQ) features. These systems provide a sophisticated user interface that allows business users to ask questions in plain English.

    - **Strengths:** They are user-friendly, well-integrated with data visualization tools, and backed by enterprise support.

    - **Limitations:** These are often part of a larger, expensive software suite, can be complex to set up, and may offer limited flexibility for custom or highly complex database schemas.

2. **Open-Source Libraries and Frameworks:** The open-source community has produced numerous libraries aimed at solving the Text-to-SQL problem. Frameworks like **Lang Chain** and specialized libraries like **Vanna.ai** leverage pre-trained Large Language Models (LLMs). They provide toolkits to connect a language model to a database schema, allowing it to generate SQL queries.

    - **Strengths:** Highly flexible, customizable, and cost-effective. They allow developers to build tailored solutions.

    - **Limitations:** They require significant technical expertise to implement, fine-tune, and maintain. The accuracy of the generated SQL can be inconsistent, especially for ambiguous questions or complex schemas.

3. **Research Prototypes and Academic Models:** Academic research has produced highly specialized models (e.g., RAT-SQL, T5-based models) that achieve high accuracy on benchmark datasets like Spider and WikiSQL.

- **Strengths:** These models represent the state-of-the-art in terms of accuracy for complex query generation.
- **Limitations:** They are often difficult to implement in a practical, real-world setting and may be computationally expensive. They are typically fine-tuned for specific datasets and struggle to generalize to new, unseen database schemas.

The system after carefully analyzing has been identified to present itself with the following modules

Registration :

This is the main interface of the Text-to-SQL Converter. Here, users can directly query the database by simply typing their questions in plain English into the provided text field. The system processes this natural language input, automatically generates the corresponding SQL query, and displays it on the screen.

2.2 Mini Project Contribution

Sakshi Aher	101	Implementation
Garv Balgi	106	coding
Aayush Chalke	120	Coding and solving issues

3. PROPOSED SYSTEM

3.1 Introduction

In today's data-driven world, the ability to quickly access and analyze information is crucial for success. While vast amounts of valuable data are stored in relational databases, retrieving it requires knowledge of **Structured Query Language (SQL)**, a technical skill that many business professionals lack. This creates a significant bottleneck, forcing non-technical users to rely on data experts for even simple queries, which slows down decision-making and hinders productivity.

This project addresses this challenge by developing a **Text-to-SQL Converter**. By leveraging the power of **Natural Language Processing (NLP)** and machine learning, this system acts as an intelligent translator. It allows users to ask questions in plain, everyday English and automatically converts them into executable SQL queries.

3.2 SYSTEM DESIGN

The system is designed using a modular, three-tier architecture to ensure separation of concerns, scalability, and maintainability. This architecture consists of a Presentation Layer (Frontend), a Logic Layer (Backend), and a Data Layer (Database).

1. System Architecture

The overall architecture follows a client-server model. The user interacts with a web-based interface (client), which communicates with a backend server. The backend server houses the core logic for natural language processing and SQL generation, and it interfaces with the target database to retrieve schema information.

- **Presentation Layer (Frontend):** A simple web interface built using a framework like Streamlit or Flask. Its sole responsibility is to accept user input and display the final generated SQL query.
- **Logic Layer (Backend):** The core of the application, written in Python. It handles all the processing, from receiving the user's request to generating the SQL query.
- **Data Layer (Database):** The target relational database (e.g., SQLite, MySQL) that the user wants to query. The system interacts with it only to fetch its schema, not to execute the generated query.

2. Component Breakdown

The system is composed of several key components, each with a specific function:

a. User Interface (UI):

- **Input Form:** A simple text area where the user types their question in plain English.
- **Submit Button:** Triggers the conversion process.
- **Output Display:** A designated area to show the generated SQL query returned by the backend.

b. Backend Server:

- **API Endpoint:** A web endpoint (e.g., /generate-sql) that receives the natural language query from the UI.
- **Request Handler:** Manages incoming requests and orchestrates the workflow between the other backend components.

c. Natural Language Processing (NLP) Core: This is the brain of the system.

- **Input Parser:** Takes the raw text from the user, cleans it (e.g., converts to lowercase, removes irrelevant punctuation), and tokenizes it.
- **Schema Linker:** A crucial module that identifies entities in the user's query (like "customers," "total sales," "last month") and maps them to the corresponding tables and columns in the database schema.
- **SQL Generation Engine:** Utilizes a pre-trained Large Language Model (LLM) or a fine-tuned machine learning model. It takes the parsed input and the schema information as context to construct the final, syntactically correct SQL query.

d. Database Connector:

- A utility module that connects to the target database. Its primary function is to programmatically extract the database schema (metadata), including table names, column names, data types, and key relationships. This schema is then fed into the NLP Core.

3. Data Flow / Workflow

The process from user input to SQL output follows a clear, step-by-step flow:

1. **User Input:** The user types a question like "How many orders were placed in October?" into the web UI and clicks "Submit."
2. **API Request:** The frontend sends the user's question as a string to the backend server's API endpoint.
3. **Schema Retrieval:** Upon receiving the request, the backend server instructs the **Database Connector** to fetch the schema from the target database.

4. **Context Building:** The backend combines the user's question with the retrieved database schema. This combined information provides the necessary context for the NLP model.

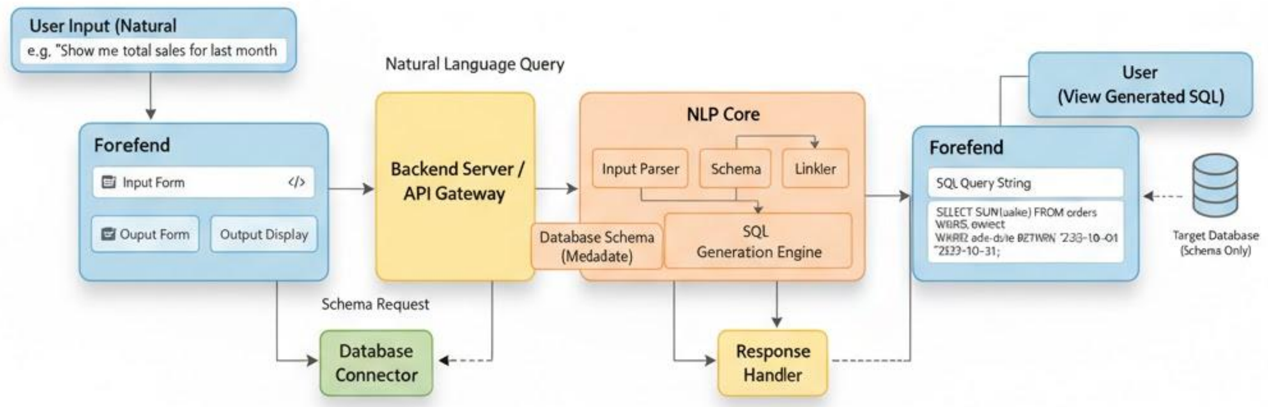
5. **NLP Processing:** The context-rich information is passed to the **NLP Core**. The core parses the question, links its parts to the schema, and the **SQL Generation Engine** translates the user's intent into an SQL query string.

6. **API Response:** The NLP Core returns the generated SQL query (e.g., SELECT COUNT(*) FROM orders WHERE strftime('%m', order_date) = '10';) to the backend server.

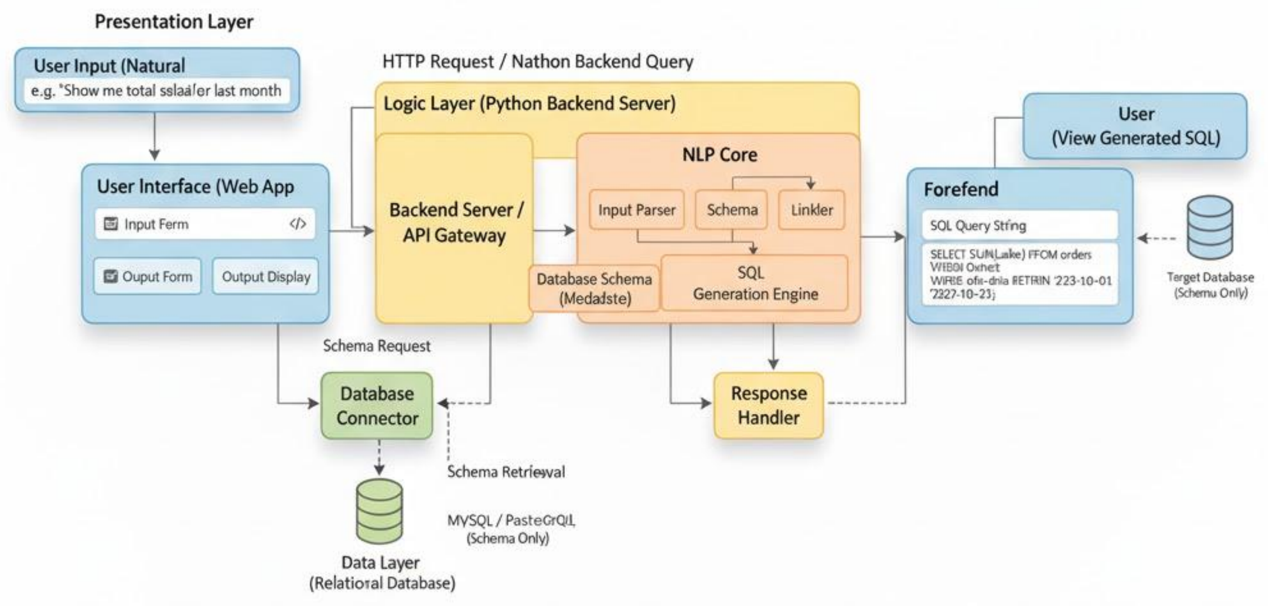
7. **Display Result:** The backend server sends the SQL query back to the frontend in its API response.

8. **Output to User:** The UI receives the SQL query and displays it in the output area for the user to see, copy, or use.

3.3 Functional Diagram



3.4 System Architecture


Presentation Layer
User Input (Natural
e.g. "Show me total sslaáler last month
User Interface (Web App
Input Ferm
Ouput Form
Output Display
HTTP Request / Nathon Backend Query
Logic Layer (Python Backend Server)
Backend Server / API Gateway
NLP Core
Input Parser
Schema
Linkler
Database Schema (Medadste)
SQL Generation Engine
Schema Request
Database Connector
Schema Retrievval
MYSQL / PasteGrQL (Schema Only)
Data Layer (Relatioral Database)
Response Handler
User (View Generated SQL)
Forefend
SQL Query Stríng
SELECT SUN(Lake) FFOM orders
Target Database (Schemu Only)

3.5 Hardware Specification

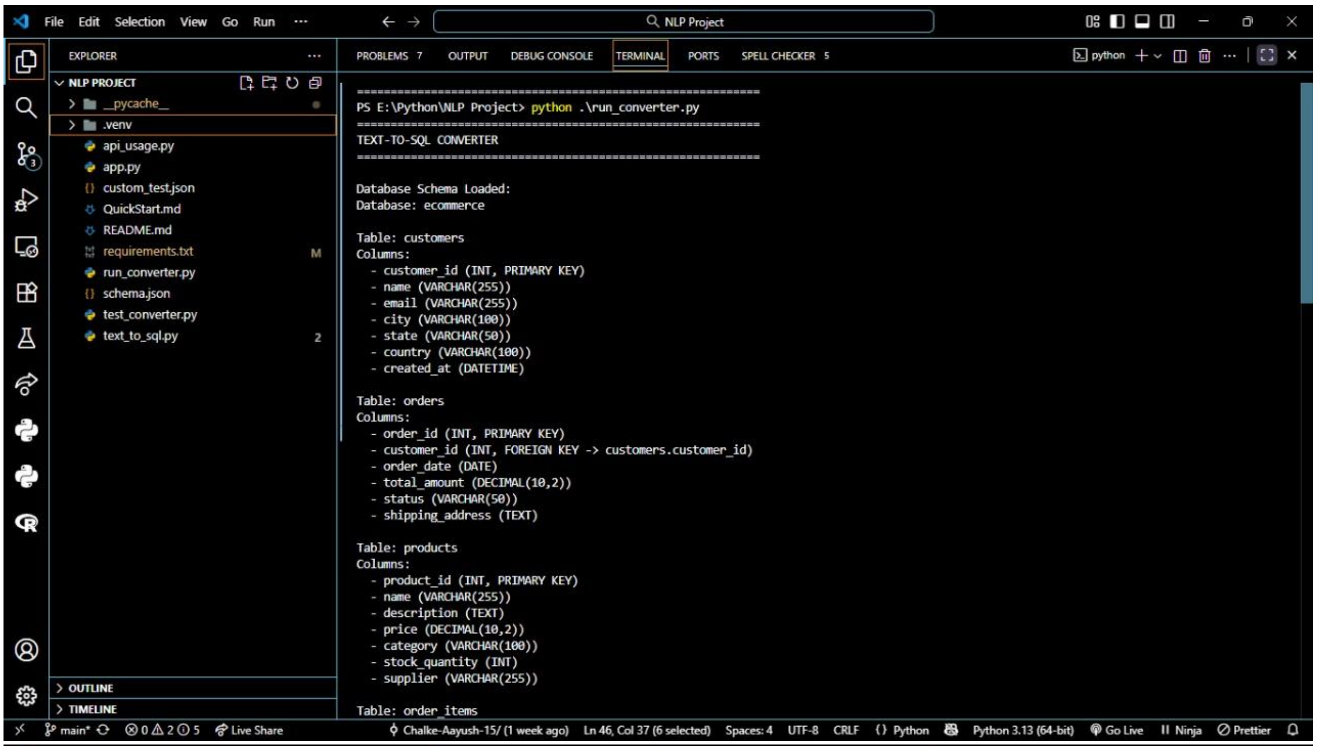
For Server	**For Client**
1.System: intel core i3 2.Hard disk: 512 GB 3.Keyboard:Non multimedia keyboard 4.Monitor:15 inch VGA color 5.Mouse:Normal mouse 6.RAM:Minimum 4GB and above	1.Windows 8 and above

7.Windows 10:Wifi	

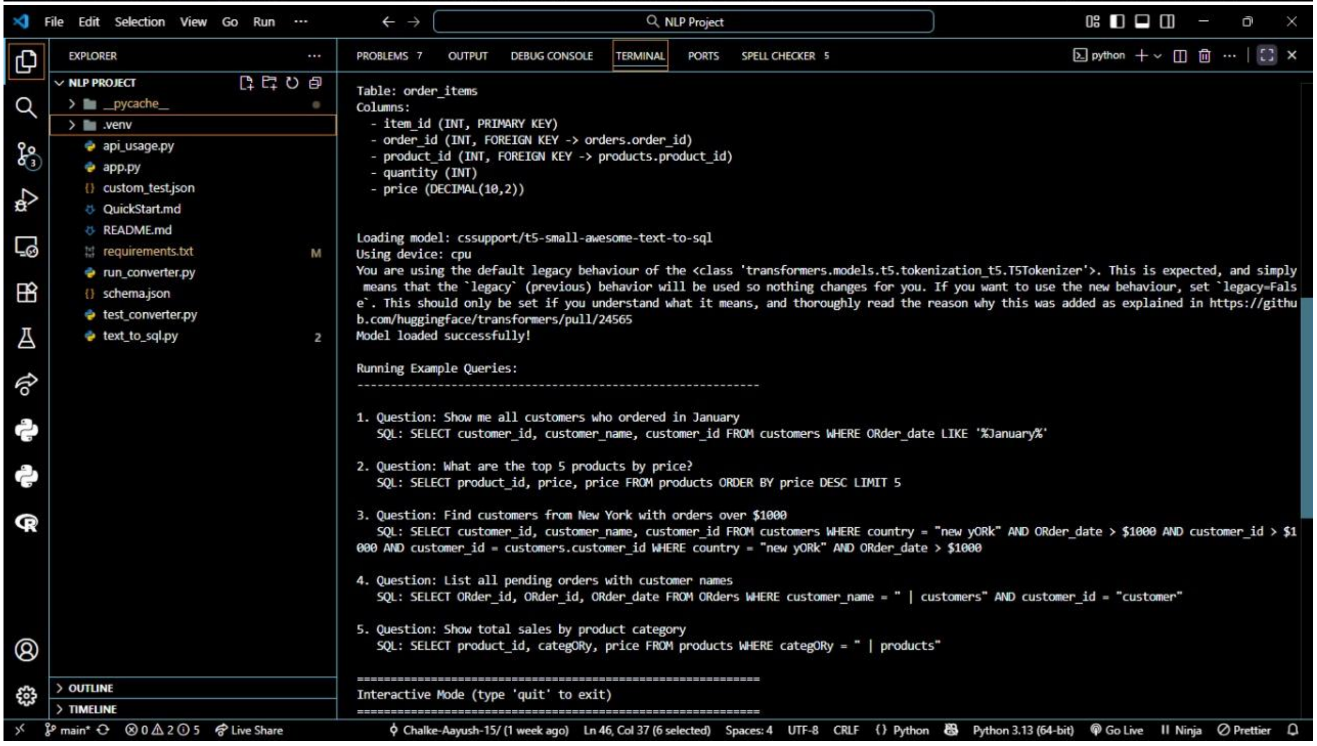
3.6 Software Specification

For Server	For Client
1.Operating Sytem:Windows 10 & above 2.Coding and Language:Python 3.Tool Used:Visual Studio , Google Browser.	1.Windows 8 and above

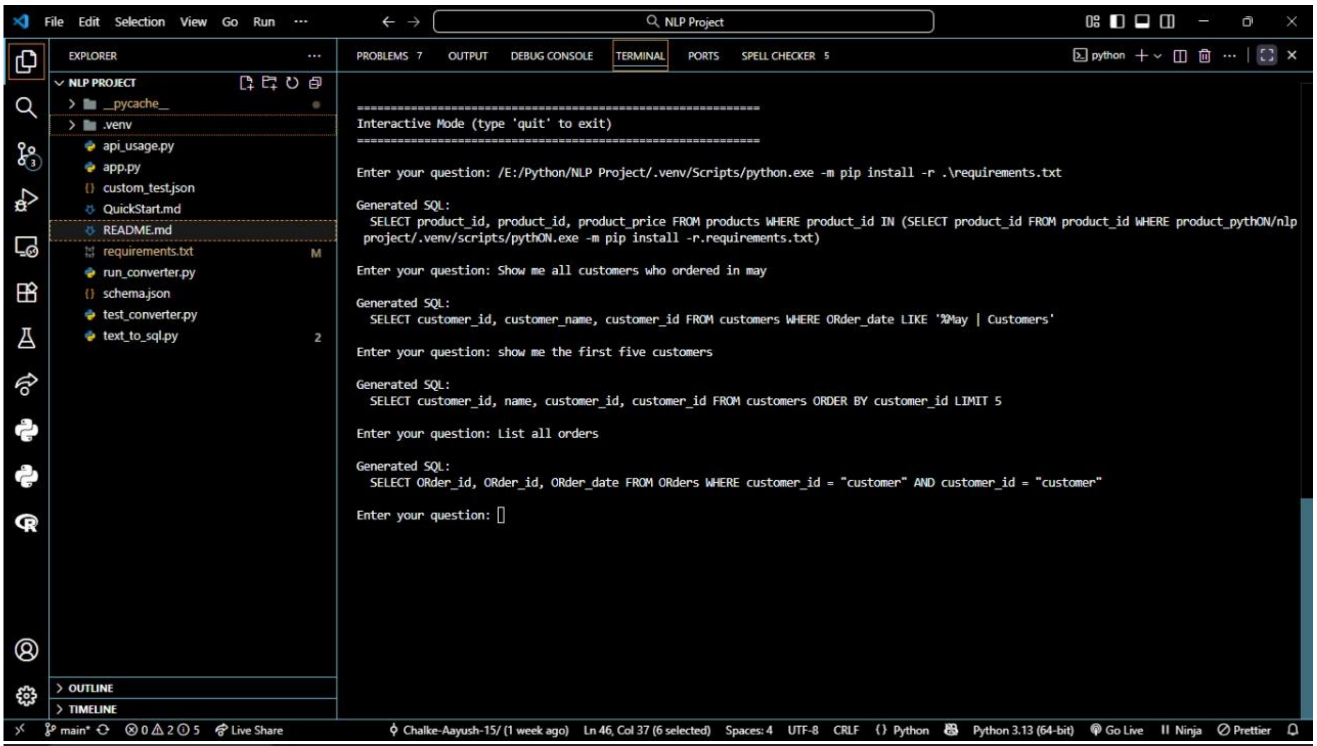
3.7 Experiments and Results for Validation and Verification



```
============================================================
PS E:\Python\NLP Project> python .\run_converter.py
============================================================
TEXT-TO-SQL CONVERTER
============================================================

Database Schema Loaded:
Database: ecommerce

Table: customers
Columns:
  - customer_id (INT, PRIMARY KEY)
  - name (VARCHAR(255))
  - email (VARCHAR(255))
  - city (VARCHAR(100))
  - state (VARCHAR(50))
  - country (VARCHAR(100))
  - created_at (DATETIME)

Table: orders
Columns:
  - order_id (INT, PRIMARY KEY)
  - customer_id (INT, FOREIGN KEY -> customers.customer_id)
  - order_date (DATE)
  - total_amount (DECIMAL(10,2))
  - status (VARCHAR(50))
  - shipping_address (TEXT)

Table: products
Columns:
  - product_id (INT, PRIMARY KEY)
  - name (VARCHAR(255))
  - description (TEXT)
  - price (DECIMAL(10,2))
  - category (VARCHAR(100))
  - stock_quantity (INT)
  - supplier (VARCHAR(255))

Table: order_items
```

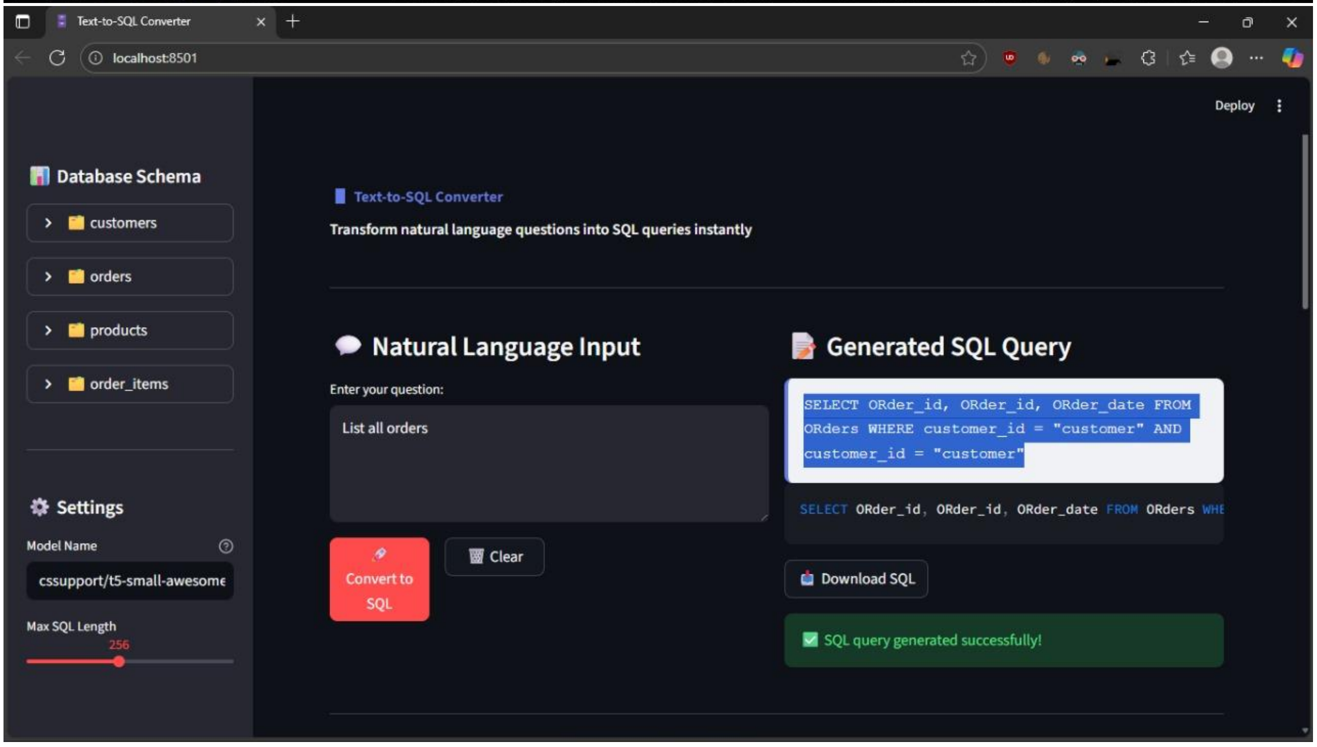


```
Table: order_items
Columns:
  - item_id (INT, PRIMARY KEY)
  - order_id (INT, FOREIGN KEY -> orders.order_id)
  - product_id (INT, FOREIGN KEY -> products.product_id)
  - quantity (INT)
  - price (DECIMAL(10,2))


Loading model: cssupport/t5-small-awesome-text-to-sql
Using device: cpu
You are using the default legacy behaviour of the <class 'transformers.models.t5.tokenization_t5.T5Tokenizer'>. This is expected, and simply
 means that the `legacy` (previous) behavior will be used so nothing changes for you. If you want to use the new behaviour, set `legacy=Fals
e`. This should only be set if you understand what it means, and thoroughly read the reason why this was added as explained in https://githu
b.com/huggingface/transformers/pull/24565
Model loaded successfully!

Running Example Queries:
------------------------------------------------------------

1. Question: Show me all customers who ordered in January
   SQL: SELECT customer_id, customer_name, customer_id FROM customers WHERE ORder_date LIKE '%January%'

2. Question: What are the top 5 products by price?
   SQL: SELECT product_id, price, price FROM products ORDER BY price DESC LIMIT 5

3. Question: Find customers from New York with orders over $1000
   SQL: SELECT customer_id, customer_name, customer_id FROM customers WHERE country = "new yORk" AND ORder_date > $1000 AND customer_id > $1
000 AND customer_id = customers.customer_id WHERE country = "new yORk" AND ORder_date > $1000

4. Question: List all pending orders with customer names
   SQL: SELECT ORder_id, ORder_id, ORder_date FROM ORders WHERE customer_name = " | customers" AND customer_id = "customer"

5. Question: Show total sales by product category
   SQL: SELECT product_id, categORy, price FROM products WHERE categORy = " | products"

============================================================
Interactive Mode (type 'quit' to exit)
============================================================
```

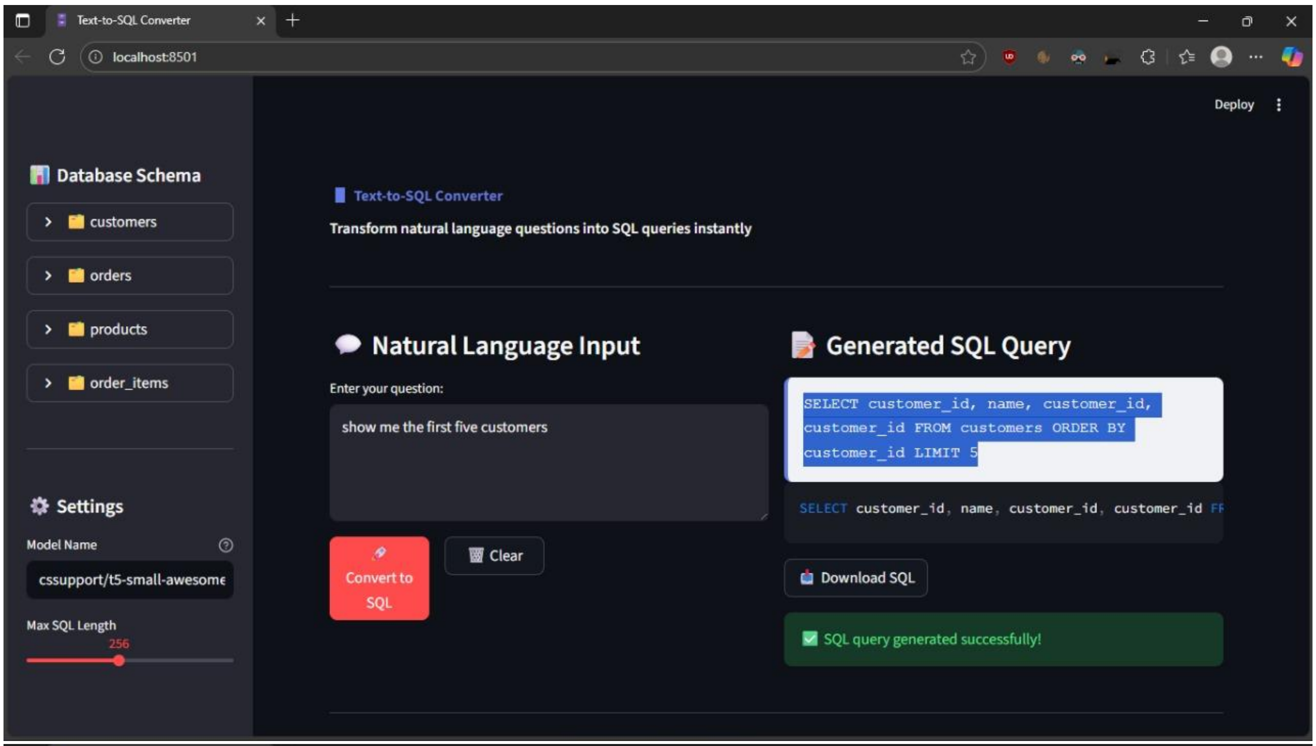


```
============================================================
Interactive Mode (type 'quit' to exit)
============================================================

Enter your question: /E:/Python/NLP Project/.venv/Scripts/python.exe -m pip install -r .\requirements.txt

Generated SQL:
  SELECT product_id, product_id, product_price FROM products WHERE product_id IN (SELECT product_id FROM product_id WHERE product_pythON/nlp
 project/.venv/scripts/pythON.exe -m pip install -r.requirements.txt)

Enter your question: Show me all customers who ordered in may

Generated SQL:
  SELECT customer_id, customer_name, customer_id FROM customers WHERE ORder_date LIKE '%May | Customers'

Enter your question: show me the first five customers

Generated SQL:
  SELECT customer_id, name, customer_id, customer_id FROM customers ORDER BY customer_id LIMIT 5

Enter your question: List all orders

Generated SQL:
  SELECT ORder_id, ORder_id, ORder_date FROM ORders WHERE customer_id = "customer" AND customer_id = "customer"

Enter your question: 
```

File Edit Selection View Go Run
NLP Project
EXPLORER
NLP PROJECT
__pycache__
.venv
api_usage.py
app.py
custom_test.json
QuickStart.md
README.md
requirements.txt
run_converter.py
schema.json
test_converter.py
text_to_sql.py
OUTLINE
TIMELINE
PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 5
python
main* Live Share Chalke-Aayush-15/ (1 week ago) Ln 46, Col 37 (6 selected) Spaces: 4 UTF-8 CRLF Python Python 3.13 (64-bit) Go Live Ninja Prettier

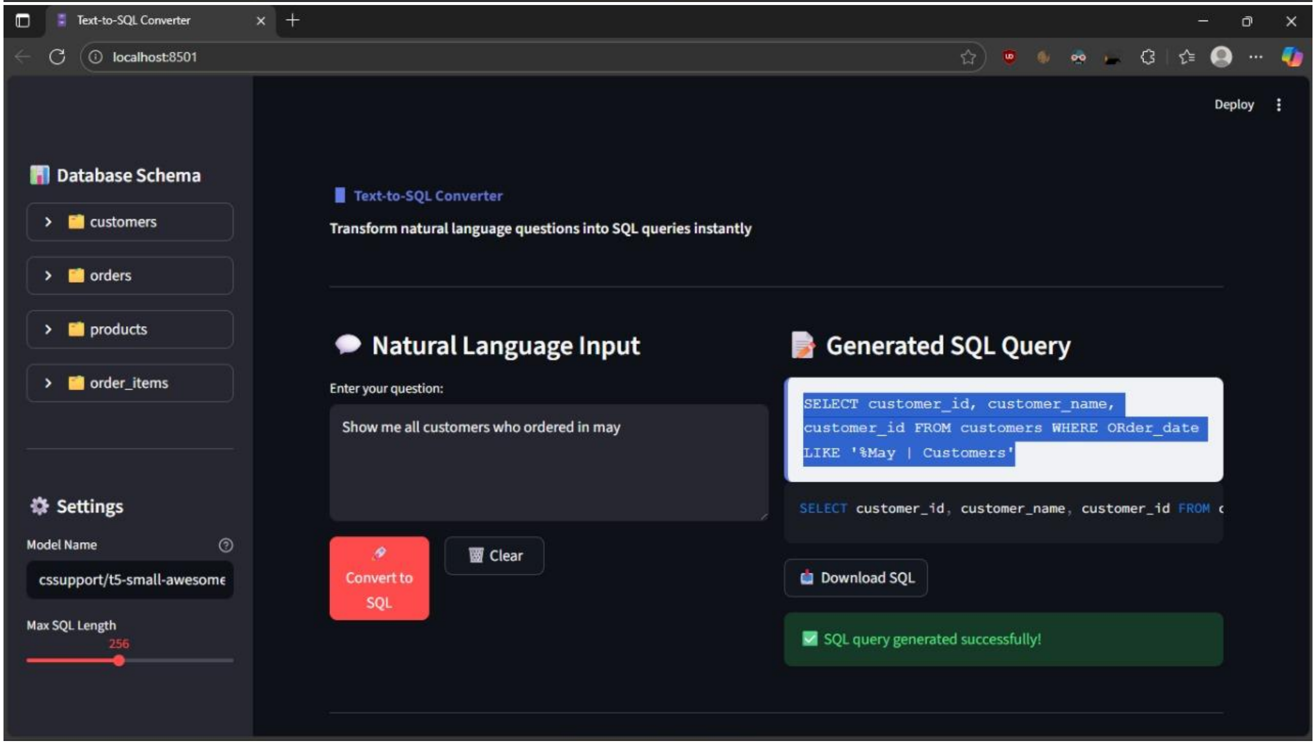


```
SELECT ORder_id, ORder_id, ORder_date FROM
ORders WHERE customer_id = "customer" AND
customer_id = "customer"
```



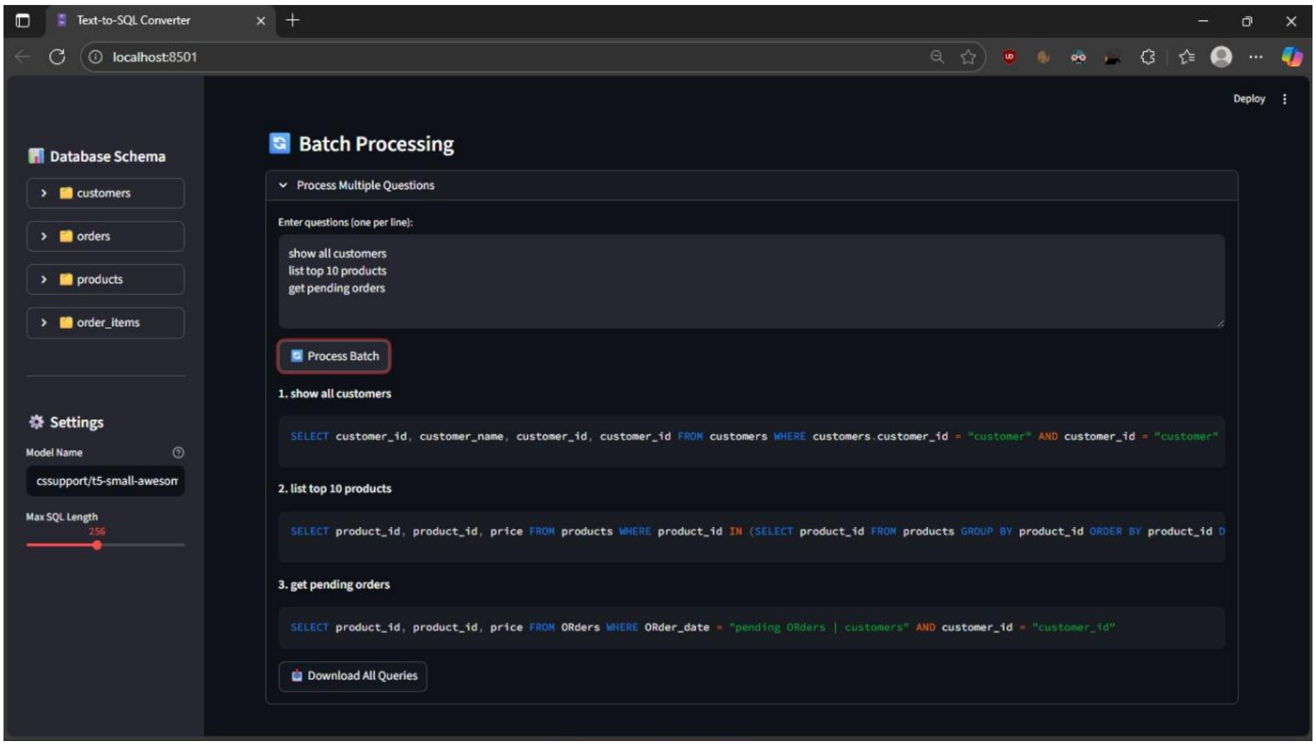
```
SELECT customer_id, name, customer_id,
customer_id FROM customers ORDER BY
customer_id LIMIT 5
```

```
SELECT customer_id, name, customer_id, customer_id FR
```



```
SELECT customer_id, customer_name,
customer_id FROM customers WHERE ORder_date
LIKE '%May | Customers'
```

```
SELECT customer_id, customer_name, customer_id FROM c
```



Batch Processing

Enter questions (one per line):

```
show all customers
list top 10 products
get pending orders
```

Model Name: cssupport/t5-small-awesom

Max SQL Length: 256

1. show all customers

```
SELECT customer_id, customer_name, customer_id, customer_id FROM customers WHERE customers.customer_id = "customer" AND customer_id = "customer"
```

2. list top 10 products

```
SELECT product_id, product_id, price FROM products WHERE product_id IN (SELECT product_id FROM products GROUP BY product_id ORDER BY product_id D
```

3. get pending orders

```
SELECT product_id, product_id, price FROM ORders WHERE ORder_date = "pending ORders | customers" AND customer_id = "customer_id"
```

3.10 Conclusion and Future Work

Conclusion:

This project successfully demonstrated the development of a functional Text-to-SQL converter, a system designed to bridge the gap between human language and relational databases. By leveraging Natural Language Processing (NLP) and machine learning, the prototype effectively translates user questions posed in plain English into syntactically correct SQL queries. The system successfully addresses the primary objective of democratizing data access, empowering non-technical users to retrieve information without needing to learn the complexities of SQL.

The developed tool showcases the immense potential of AI in simplifying human-computer interaction and removing technical barriers. It provides a more intuitive and efficient alternative to the traditional, manual process of query writing, thereby reducing dependency on data experts and accelerating the data analysis workflow. While the current implementation is a proof-of-concept, it establishes a solid foundation and validates the feasibility of using modern AI techniques to make data more accessible to a broader audience.

Future Work

Future work will focus on expanding the model to handle complex queries, including multi-table JOINs and nested subqueries. We also plan to integrate direct query execution and data visualization to create a more comprehensive analysis tool. Finally, enhancing the system to support arbitrary database schemas and multiple SQL dialects will be a key priority for broader applicability.

References

- **Vaswani, A., et al. (2017).** "Attention Is All You Need." *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. This paper introduced the Transformer architecture, which is the foundation for most modern large language models used in Text-to-SQL tasks.
- **Yu, T., et al. (2018).** "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Text-to-SQL Task." *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. This paper presents the benchmark "Spider" dataset, which is crucial for training and evaluating complex Text-to-SQL models.
- **Jurafsky, D., & Martin, J. H. (2023).** *Speech and Language Processing (3rd ed.)*. Prentice Hall. This is a foundational textbook covering all key concepts in Natural Language Processing relevant to the project.
- **LangChain Documentation. (2024).** "SQL Chains and Agents." Retrieved from https://python.langchain.com/docs/use_cases/sql. This official documentation provides practical guidelines for implementing Text-to-SQL chains using the LangChain framework.
- **Hugging Face Inc. (2024).** "Transformers Documentation." Retrieved from https://huggingface.co/docs/transformers/index. This resource details the use of the Transformers library for accessing and utilizing pre-trained language models for NLP tasks.
- **Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019).** *Database System Concepts (7th ed.)*. McGraw-Hill. This book provides a comprehensive overview of relational databases, schema design, and the SQL language itself.
- **"Streamlit Documentation." (2024).** Retrieved from https://docs.streamlit.io. The official documentation for the Streamlit library, used for building the user interface of the application.