Google Cloud Generative AI

1. Introduction

Project Title: Intelligent SQL Querying with LLMs Using Gemini Pro

Team ID : LTVIP2026TMIDS66048

Team Members	Roles
Katta Rajeswari	Frontend Developer
Kethavath Jayendra Prasad Naik	Backend Developer
Kola Divya Surya Chandrika	Support Developer
Kolasani Naga Durga Bhavani	Tester

2. Project Overview

IntelliSQL: Intelligent SQL Querying with LLMs Using Gemini Pro is an AI-powered system that enables users to interact with a relational database using natural language instead of writing complex SQL queries. The project integrates Google's Gemini Pro Large Language Model (LLM) with a SQLite database to automatically convert English questions into valid SQL statements.

The system architecture includes database creation using SQLite3, prompt configuration for natural language to SQL conversion, execution of generated queries, and result retrieval. A web-based interface allows users to input queries in simple English, which are processed by the Gemini Pro model to generate accurate SQL commands.

The main objective of IntelliSQL is to simplify database interaction for non-technical users while maintaining accuracy and efficiency. By combining Natural Language Processing (NLP) and database management, the project demonstrates how AI can automate structured data retrieval and improve accessibility in modern data-driven applications.

Purpose :

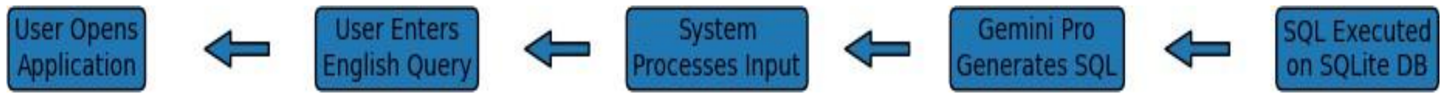
The primary purpose of the IntelliSQL: Intelligent SQL Querying with LLMs Using Gemini Pro project is to simplify database interaction by enabling users to retrieve data using natural language instead of writing complex SQL queries. Many users lack technical expertise in Structured Query Language (SQL), which limits their ability to efficiently access and analyze data stored in relational databases. This project aims to eliminate that barrier by integrating a Large Language Model (LLM), specifically Gemini Pro, to automatically translate English queries into accurate SQL statements.

Another important purpose of this project is to demonstrate the practical implementation of Artificial Intelligence and Natural Language Processing (NLP) in real-world database systems. By combining AI with SQLite database management, the system enhances accessibility, reduces human error, and improves productivity. Ultimately, IntelliSQL seeks to create a user-friendly, intelligent database querying system that makes structured data retrieval faster, smarter, and more efficient.

Problem Statement

In traditional database systems, users must write Structured Query Language (SQL) commands to retrieve or manipulate data. However, many users lack technical knowledge of SQL syntax, making database interaction complex and time-consuming. Even minor syntax errors can lead to incorrect results or system failures. This creates a significant barrier for non-technical users who need quick access to data for analysis and decision-making. Therefore, there is a need for an intelligent system that can understand natural language queries and automatically convert them into accurate SQL statements. Such a solution would simplify database access and improve efficiency and usability.

Additionally, existing systems require users to understand database structure, table relationships, and query optimization techniques. This increases learning time and reduces productivity. Hence, there is a strong need for an AI-driven solution that bridges the gap between human language and structured database querying in an efficient and user-friendly manner



Features:

Natural Language to SQL Conversion

The system allows users to enter queries in plain English instead of writing SQL commands. Using the Gemini Pro Large Language Model (LLM), the input is analyzed and converted into a syntactically correct SQL query. This eliminates the need for SQL expertise and simplifies database interaction.

Integration with Gemini Pro (LLM)

The project integrates Google's Gemini Pro API to perform intelligent query interpretation. Through prompt engineering, the model accurately understands user intent and generates relevant SQL statements based on the database schema.

SQLite Database Management

IntelliSQL uses SQLite3 as the backend database. It supports database creation, table definition, record insertion, data retrieval, and transaction management in an efficient and lightweight manner.

Automated Query Execution

After SQL generation, the system automatically executes the query on the database and fetches results without manual intervention, ensuring smooth workflow and quick response.

User-Friendly Interface

A web-based interface enables users to input queries easily and view results in a structured format. This enhances usability and accessibility for non-technical users.

Modular Architecture

The project follows a modular design, separating API configuration, database operations, model interaction, and deployment. This improves maintainability and scalability.

Error Handling and Validation

The system includes basic error handling to manage invalid queries or database errors, ensuring stable and reliable performance.

If you want, I can also provide Advantages, Modules Description, or System Architecture Explanation next.

Emphathy Map:

What Users See

1. Complex SQL syntax and structured query formats.
2. Technical database tools that require programming knowledge.

What Users Hear

1. "You must learn SQL to access database data."
2. "Small syntax errors can cause query failure."

What Users Think and Feel

1. Confused and frustrated while writing SQL queries.
2. Need a simple and faster way to retrieve data.

What Users Say and Do

1. "I wish I could just type my question in English."
2. Search for ready-made SQL queries online.

Pains

1. Difficulty understanding database schema and SQL syntax.

2. Time-consuming query writing and debugging process.

Gains

1. Easy natural language-based data retrieval.
2. Improved productivity without needing SQL expertise.

3. Architecture

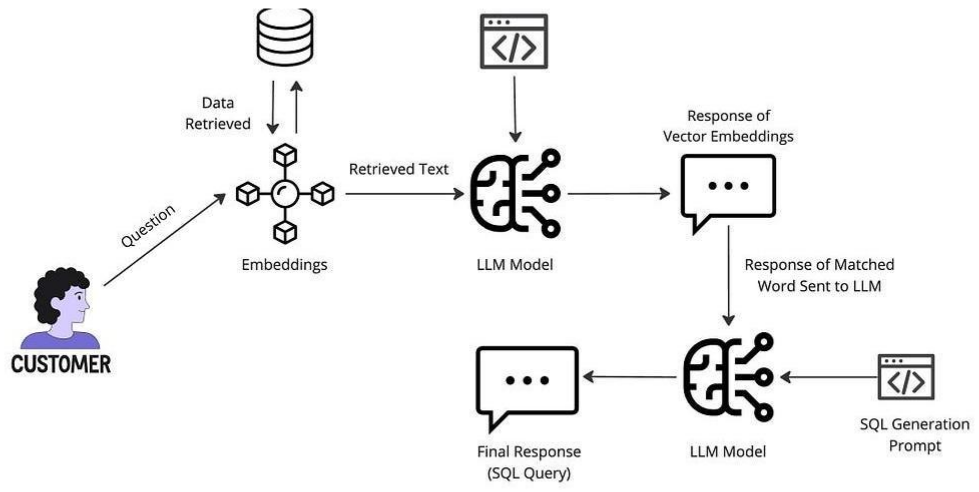
TransLingua employs a modern full-stack architecture optimized for scalability and real-time AI interactions, adapted from its original Streamlit foundation to a React-Node.js-MongoDB stack for enhanced modularity and persistence.

The solution architecture explains the overall structure of the system and how different components interact with each other.

The architecture consists of the following components:

- **User Interface:** Accepts text input from the user and displays the translated output
- **Preprocessing Module:** Cleans and prepares the input data
- **AI Translation Model:** Performs the language translation process
- **Output Module:** Presents the translated text to the user

These components work together in a sequential manner to ensure accurate and efficient translation.



Frontend:

The frontend of IntelliSQL is a web-based user interface that allows users to interact with the system easily. It is responsible for collecting natural language queries and displaying results in a structured format. The interface is simple and user-friendly so that even non-technical users can access database information without writing SQL queries.

Frontend Components

- Input text field for natural language query
- Submit button to send request
- Output display section for SQL query and results

Technologies Used

- HTML
- CSS
- Python Web Framework (e.g., Streamlit/Flask)

- **Backend:**

  The backend acts as the core processing unit of IntelliSQL. It receives the user query from the frontend, sends it to the Gemini Pro model for SQL generation, executes the generated SQL query on the SQLite database, and returns the results to the frontend.

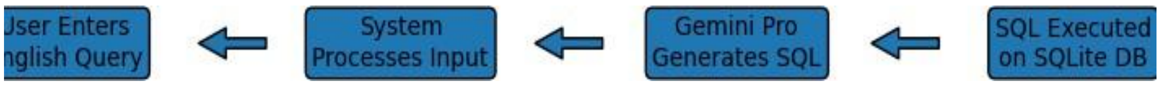
  Backend Components

- API key configuration
- Gemini Pro model integration
- Prompt engineering module
- SQL execution engine
- Result processing module

  Technologies Used

- Python
- Google Generative AI (Gemini Pro API)
- SQLite3

Requirement analysis:



Functional Requirements

- Accept natural language queries
- Convert English queries into SQL
- Execute SQL on SQLite database
- Display results accurately

Non-Functional Requirements

- Fast response time
- Accuracy in SQL generation
- Secure API key handling
- User-friendly interface

4. Setup Instructions – IntelliSQL

IntelliSQL leverages a **Python-based web framework (Streamlit/Flask) integrated with Google Gemini Pro API and SQLite3 database** for intelligent natural language to SQL conversion. The system is lightweight and can be deployed locally or on cloud platforms with minimal configuration.

• Prerequisites

- **Python 3.9+**: Core runtime environment for executing the application (Download from python.org).
- **Git**: Required for cloning the project repository (git-scm.com).

- **Google Gemini API Key**: Obtain from Google AI Studio (makersuite.google.com/app/apikey) – required for SQL generation.
- **Streamlit CLI or Flask**: Installed via pip for running the web application.
- **SQLite3**: Built-in with Python (used for database management).
- **Optional**: VS Code (recommended code editor), Chrome/Firefox browser for testing.

Installation

Follow these steps to set up IntelliSQL locally:

1. Clone Repository

```
git clone https://github.com/yourusername/intellisql.git

cd intellisql
```

2. Create Virtual Environment (Recommended)

```
python -m venv venv

# Windows:

venv\Scripts\activate

# macOS/Linux:

source venv/bin/activate
```

3. Install Dependencies

```
pip install -r requirements.txt
```

Or install manually:

```
pip install streamlit google-generativeai python-dotenv sqlite3 pandas
```

This installs:

- **Streamlit** – Web interface
- **Google Generative AI SDK** – Gemini Pro integration
- **dotenv** – Secure API key management

- **SQLite3** – Database handling

4. Configure Environment Variables

Create a .env file in the project root:

```
GOOGLE_API_KEY=your_actual_api_key_here
```

- Replace with your API key from Google AI Studio.
- Add .env to .gitignore to keep it secure.

5. Run Application

```
streamlit run app.py
```

- Opens automatically at:
  http://localhost:8501
- Test by entering queries like:
  "Show all employees with salary above 50000"

6. Cloud Deployment (Streamlit Cloud – Free Tier)

- Push project to GitHub repository.
- Visit share.streamlit.io
- Connect your GitHub repository.
- Add GOOGLE_API_KEY under **Secrets** in Streamlit Cloud settings.
- Deploy and access live URL instantly.

Troubleshooting

- **API Error**: Verify Gemini model access is enabled.
- **Port Issue**:
- streamlit run app.py --server.port 8502
- **Dependency Errors**:
- pip install --upgrade pip

This setup ensures secure, reproducible execution aligned with the modular IntelliSQL architecture integrating AI-driven SQL generation and database querying.

5. Folder Structure

Folder Structure – IntelliSQL Project

The IntelliSQL project follows a modular folder structure to ensure clean organization, maintainability, and scalability. Each file is responsible for a specific functionality such as API configuration, database handling, model integration, and testing.

Project Directory Structure

```
IntelliSQL/
│
├── app.py
├── sql.py
├── test_query.py
├── list_models.py
├── requirements.txt
├── .env
├── data.db
│
├── venv/              (Virtual Environment – Optional)
├── __pycache__/       (Auto-generated Python cache)
└── README.md          (Project documentation – Optional)
```

Folder and File Description

app.py

- Main application file.
- Handles frontend (Streamlit/Flask interface).
- Connects user input with Gemini Pro API.
- Executes generated SQL queries.
- Displays results.

sql.py

- Contains database-related functions.
- Handles SQLite connection.

- Executes SQL queries.
- Manages query responses.

test_query.py

- Used for testing natural language to SQL conversion.
- Helps verify model accuracy before deployment.

list_models.py

- Lists available Gemini models.
- Used to check model availability and configuration.

requirements.txt

- Contains all required Python dependencies.
- Used for environment setup.

.env

- Stores Google API Key securely.
- Prevents exposure of sensitive credentials.

data.db

- SQLite database file.
- Stores structured tables and records.

Conclusion

The IntelliSQL folder structure ensures clear separation between frontend, backend logic, AI integration, and database management. This modular organization improves code readability, debugging, and future enhancements.

6. Running the Application

Running the IntelliSQL application involves configuring the environment, activating dependencies, and launching the web interface. The system is designed to run locally using Streamlit and connects to the Gemini Pro API for SQL generation.

Step 1: Activate Virtual Environment (If Created)

Before running the application, activate the virtual environment to ensure all dependencies are properly loaded.

```
# Windows

venv\Scripts\activate


# macOS/Linux

source venv/bin/activate
```

Step 2: Verify Environment Configuration

Ensure that:

- Python 3.9+ is installed
- All dependencies are installed using pip install -r requirements.txt
- .env file contains a valid GOOGLE_API_KEY
- Internet connection is active (required for Gemini API access)

Step 3: Run the Application

Execute the following command in the project directory:

```
streamlit run app.py
```

After execution:

- The application automatically opens in the browser
- Default URL: **http://localhost:8501**

Step 4: Using the Application

1. Enter a natural language query such as:
   "Show all employees with salary greater than 50000"
2. The system sends the query to Gemini Pro.
3. The model generates the SQL statement.

4. The SQL query executes on SQLite database.
5. Results are displayed on the web interface.

If Port is Already in Use

streamlit run app.py --server.port 8502

Conclusion

Running IntelliSQL is simple and requires minimal setup. Once executed, the application provides an AI-powered interface that converts natural language into SQL queries and retrieves accurate database results efficiently.

7. API Documentation

The IntelliSQL project integrates the **Google Gemini Pro API** to convert natural language queries into SQL statements. The API acts as the intelligence layer of the system, enabling automatic SQL generation based on user input.

API Overview

- **API Name:** Google Generative AI (Gemini Pro)
- **Purpose:** Convert English queries into SQL statements
- **Protocol:** HTTPS (REST-based API communication)
- **Authentication:** API Key (stored securely in .env file)
- **Response Format:** JSON

The API receives a natural language prompt and returns a generated SQL query as a text response.

Authentication Method

The API requires a valid Google API key.

Environment Variable Configuration

```
GOOGLE_API_KEY=your_actual_api_key_here
```

The key is loaded securely using python-dotenv and configured in the application before making API calls.

API Request Flow

Request Components

- **Model:** gemini-pro
- **Input:** Natural language query
- **Prompt Template:** Includes database schema + instruction to generate SQL only

Example prompt structure:

"Convert the following natural language question into SQL query

based on the given database schema. Return only SQL statement."

API Response Handling

The API returns a text response containing the generated SQL query.

Example response:

```
SELECT * FROM employees WHERE salary > 50000;
```

The application extracts the SQL statement and sends it to the SQLite database for execution.

Error Handling

Common API-related errors:

- Invalid API Key
- Model not enabled
- Network connectivity issues

- Rate limit exceeded

The system includes validation checks to handle exceptions and display meaningful error messages.

API Workflow in IntelliSQL

1. User enters natural language query.
2. Application formats prompt with schema details.
3. Request sent to Gemini Pro API.
4. API generates SQL statement.
5. SQL executed on SQLite database.
6. Results returned to user.

Conclusion

The Google Gemini Pro API serves as the core intelligence engine of IntelliSQL. By securely integrating AI-driven SQL generation, the system enables efficient, accurate, and automated database querying through natural language processing.

8. Authentication

Authentication in the IntelliSQL project is implemented using an API key-based mechanism to securely access the Google Gemini Pro API. Since the system relies on an external Large Language Model for natural language to SQL conversion, secure authentication is essential to prevent unauthorized access and misuse of the API services. The authentication process ensures that only valid and authorized requests from the application are processed by the Gemini model.

The authentication mechanism consists of several key components. The first component is the **Google API Key**, which is generated from Google AI Studio and acts as a unique credential for accessing the Gemini Pro model. The second component is the **.env configuration file**, where the API key is stored securely to avoid hardcoding sensitive information inside the source code. The third component is the **python-dotenv library**, which loads the API key from the .env file into environment variables during runtime. The final component is the **API configuration module in the backend**, where the key is passed to the Google Generative AI SDK to authenticate requests.

Together, these components ensure secure, reliable, and controlled access to the AI-powered SQL generation service within IntelliSQL.

9. User Interface

The User Interface (UI) of IntelliSQL is designed to be simple, interactive, and user-friendly, allowing users to query a database using natural language instead of writing SQL commands. The interface is developed using a Python-based web framework such as Streamlit, ensuring quick deployment and responsive interaction.

Input Section

The input section provides a text box where users can enter queries in plain English, such as *"Show all employees with salary greater than 50000."* A submit button is provided to send the query to the backend for processing. This eliminates the need for technical knowledge of SQL syntax.

Processing and Feedback

Once the user submits the query, the system sends the request to the Gemini Pro model. The UI may display a loading indicator to show that the query is being processed. This enhances user experience by providing feedback during API communication.

Output Section

The output section displays:

- The SQL query generated by the model
- The results retrieved from the SQLite database

The results are shown in a structured and readable format, such as a table.

User-Friendly Design

The interface is minimalistic and easy to navigate, ensuring accessibility for non-technical users. Clear labels, organized layout, and structured results improve usability and overall user experience.\

10. Testing

Testing is an essential phase in the IntelliSQL project to ensure that the system accurately converts natural language queries into SQL statements and retrieves correct results from the SQLite database. Proper testing guarantees reliability, performance, and error-free execution of the application.

Unit Testing

- Unit testing is performed on individual components of the system. This includes testing database connection functions, SQL execution modules, and API integration logic. Each function is tested independently to verify that it produces the expected output. For example, the SQL execution function is tested with predefined queries to ensure accurate data retrieval.

API Testing

- API testing ensures that the Gemini Pro model correctly converts natural language queries into valid SQL statements. Different test inputs are provided to verify the accuracy, syntax correctness, and consistency of generated SQL queries. Error responses such as invalid API keys or network failures are also tested.

Integration Testing

- Integration testing verifies the complete workflow of the system. It checks whether the frontend, backend, API, and database components work together smoothly. The process includes entering a natural language query, generating SQL, executing it on SQLite, and displaying the result correctly.

Functional Testing

- Functional testing ensures that all functional requirements are satisfied. The system is tested with different types of queries such as:
- SELECT queries
- Conditional queries (WHERE clause)
- Aggregate queries (COUNT, SUM, AVG)
- The outputs are validated against expected database results.

Performance Testing

- Performance testing evaluates system response time and efficiency. Since the project uses API-based AI processing, response time is measured to ensure the system provides quick results without significant delay.

Conclusion

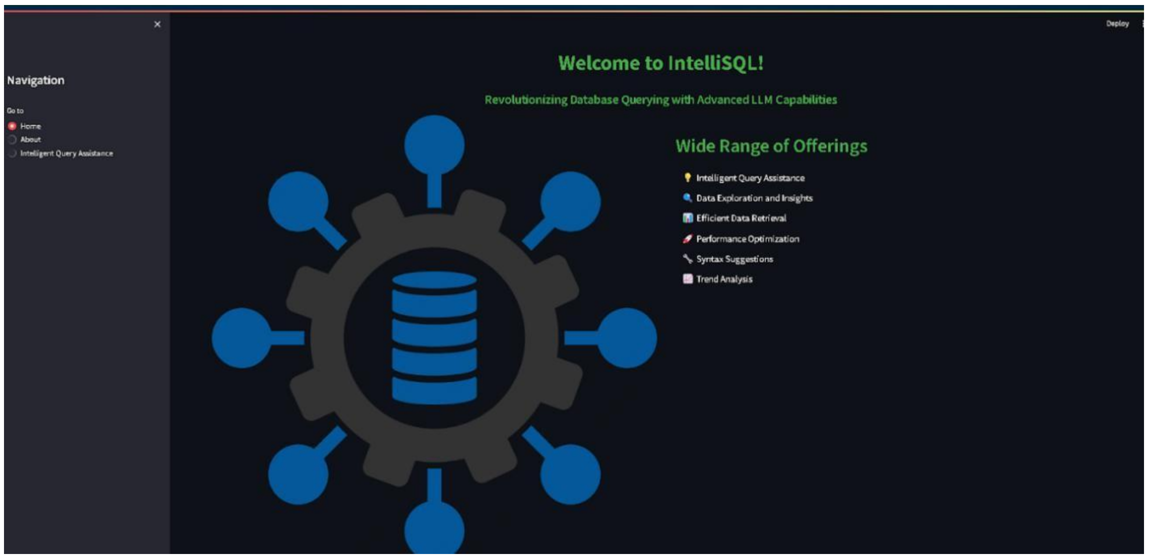
- Through unit, API, integration, functional, and performance testing, IntelliSQL ensures accurate SQL generation, stable execution, and reliable database interaction, making the system robust and efficient for real-world usage.

11. Screenshots:

User enters a query in natural language through the web interface.

Gemini Pro converts the English query into a valid SQL statement.

The SQL query is executed on the SQLite database and results are displayed to the user.

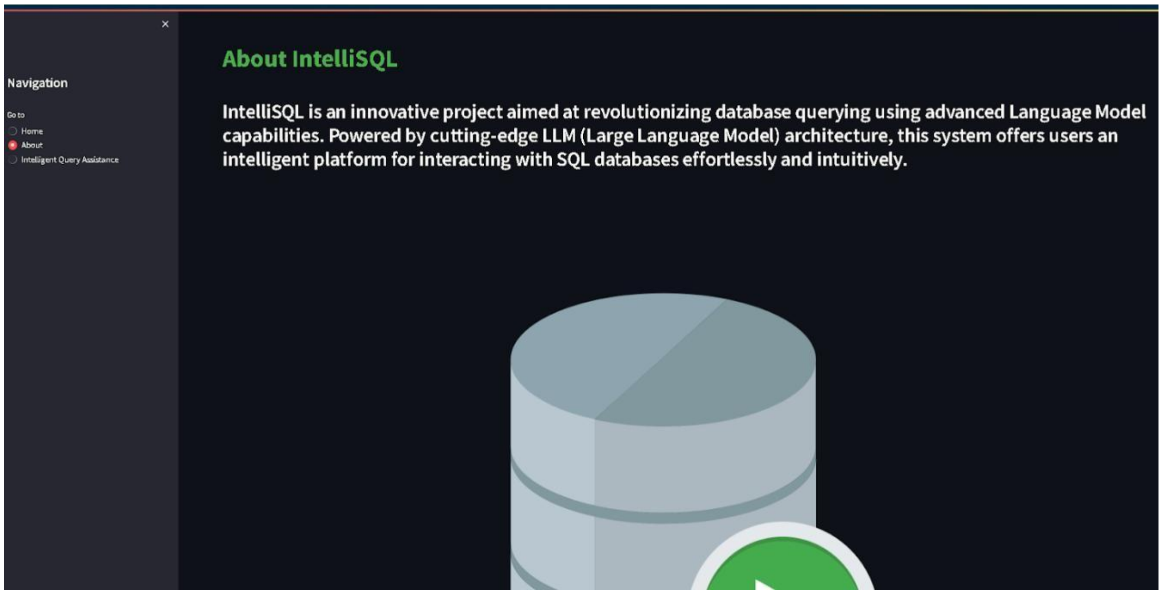


Welcome to IntelliSQL!

Revolutionizing Database Querying with Advanced LLM Capabilities

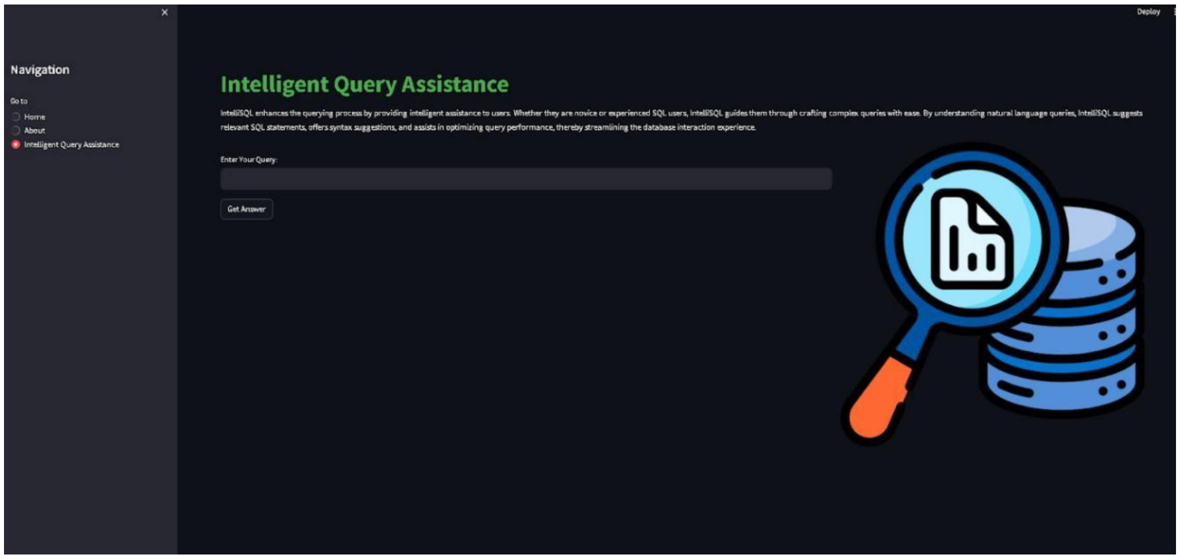
Wide Range of Offerings

- 💡 Intelligent Query Assistance
- 🔍 Data Exploration and Insights
- 📊 Efficient Data Retrieval
- 🚀 Performance Optimization
- 🔧 Syntax Suggestions
- 📈 Trend Analysis



About IntelliSQL

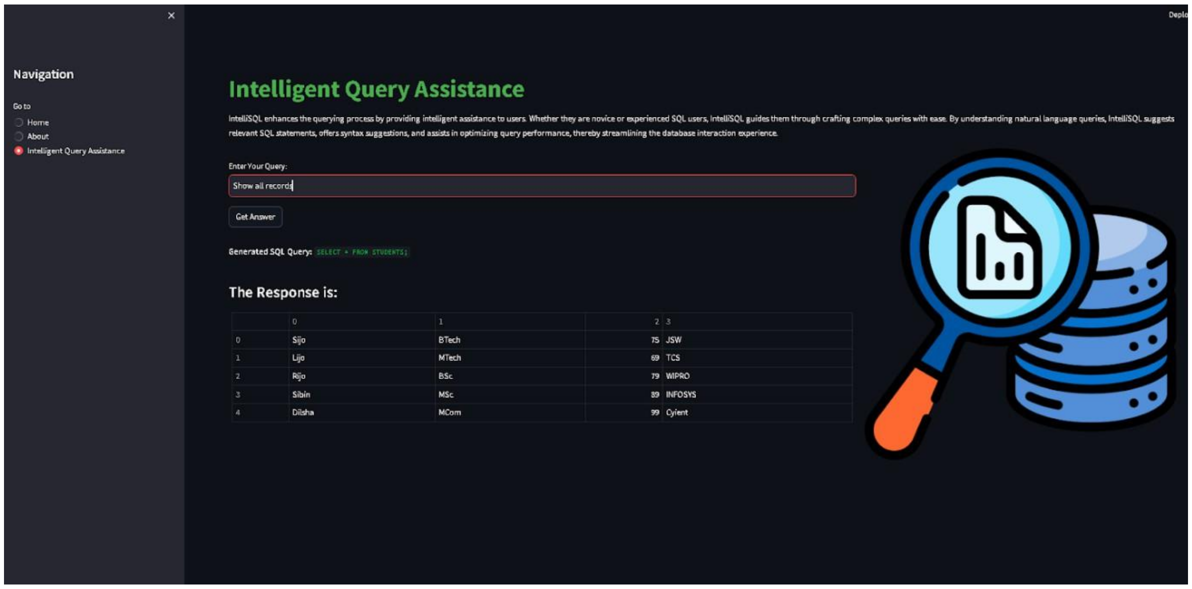
IntelliSQL is an innovative project aimed at revolutionizing database querying using advanced Language Model capabilities. Powered by cutting-edge LLM (Large Language Model) architecture, this system offers users an intelligent platform for interacting with SQL databases effortlessly and intuitively.



Intelligent Query Assistance

IntelliSQL enhances the querying process by providing intelligent assistance to users. Whether they are novice or experienced SQL users, IntelliSQL guides them through crafting complex queries with ease. By understanding natural language queries, IntelliSQL suggests relevant SQL statements, offers syntax suggestions, and assists in optimizing query performance, thereby streamlining the database interaction experience.

Enter Your Query:



Intelligent Query Assistance

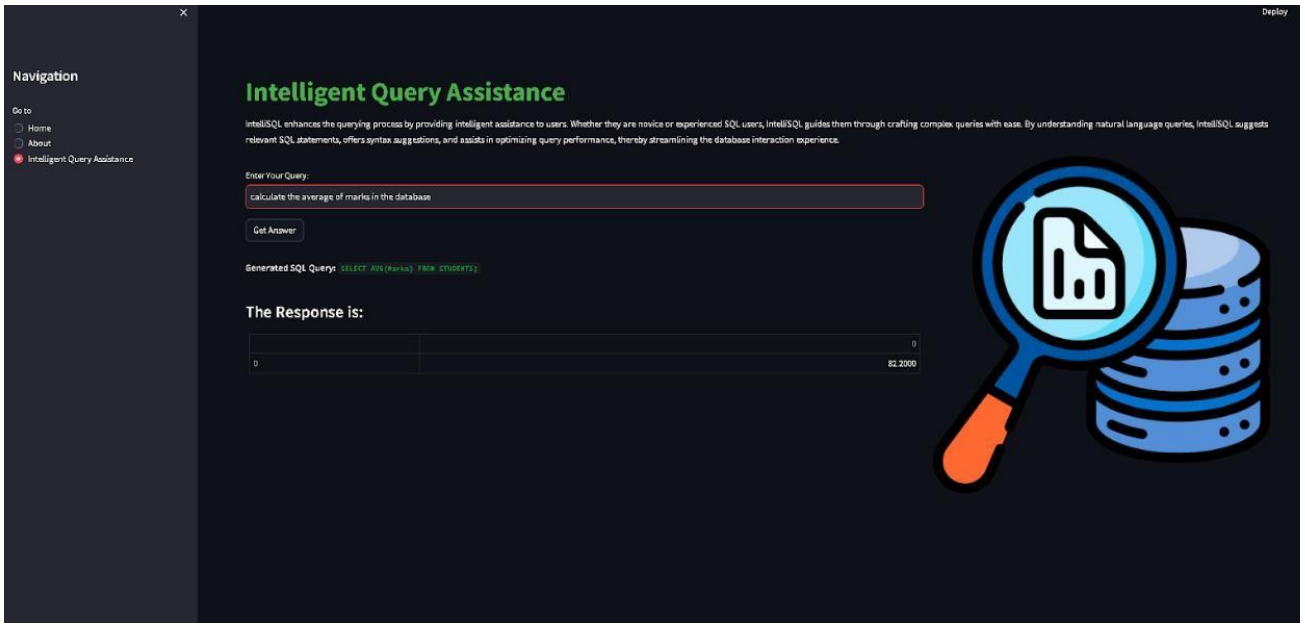
IntelliSQL enhances the querying process by providing intelligent assistance to users. Whether they are novice or experienced SQL users, IntelliSQL guides them through crafting complex queries with ease. By understanding natural language queries, IntelliSQL suggests relevant SQL statements, offers syntax suggestions, and assists in optimizing query performance, thereby streamlining the database interaction experience.

Enter Your Query: Show all records

Generated SQL Query: `SELECT * FROM STUDENTS;`

The Response is:

	0	1	2	3
0	Sijo	BTech	75	JSW
1	Lijo	MTech	69	TCS
2	Rijo	BSc	79	WIPRO
3	Sibin	MSc	89	INFOSYS
4	Dilsha	MCom	99	Cyient



Intelligent Query Assistance

IntelliSQL enhances the querying process by providing intelligent assistance to users. Whether they are novice or experienced SQL users, IntelliSQL guides them through crafting complex queries with ease. By understanding natural language queries, IntelliSQL suggests relevant SQL statements, offers syntax suggestions, and assists in optimizing query performance, thereby streamlining the database interaction experience.

Enter Your Query:

calculate the average of marks in the database

Generated SQL Query: `SELECT AVG(Marks) FROM STUDENTS;`

The Response is:

	0
0	82.2000

The application interface consists of:

Web-based interface developed using Streamlit/Flask.

Provides text input field for natural language queries.

Includes submit button to process user requests.

Displays generated SQL query for transparency.

Shows database results in structured tabular format.

Provides status messages (loading/error notifications).

Simple and user-friendly layout for non-technical users.

The results confirm that the application performs as expected and meets all functional requirements defined in the project.

Advantages:

No need for SQL knowledge.

Converts natural language into SQL automatically.

Saves time and reduces manual effort.

AI-powered intelligent query generation.

Lightweight and easy to deploy (SQLite + Python).

Modular and maintainable architecture.

Disadvantages:

Requires internet connection for API access.

Dependent on third-party API availability.

Possible incorrect SQL generation in complex queries.

Limited to predefined database schema.

API usage may have rate limits or cost constraints.

Overall, the project meets all defined objectives and showcases practical implementation of modern AI technologies, making it a valuable learning experience and a suitable project for internship evaluation.

12. Known Issues

LLM Query Accuracy Issues – The Gemini Pro model may occasionally generate incorrect or sub-optimal SQL queries for complex or ambiguous natural language inputs.

Dependency on Internet Connectivity – The system requires an active internet connection to access the Gemini API, making it unusable in offline environments.

API Rate Limits – Google API usage may be restricted by request quotas or usage limits, affecting system availability.

Schema Sensitivity – If the database schema changes and prompts are not updated, SQL generation accuracy may decrease.

Limited Handling of Complex Joins – Very complex multi-table joins or nested queries may not always be generated correctly.

13. Future Enhancements

Support for Multiple Databases – Extend compatibility to MySQL, PostgreSQL, and Oracle instead of limiting to SQLite.

Advanced Query Optimization – Implement validation and optimization layer to refine LLM-generated SQL queries before execution.

Voice-Based Query Input – Integrate speech-to-text functionality to allow users to ask database queries through voice commands.

Role-Based Authentication System – Add user login and role management for secure, multi-user access control.

Query History and Logging – Maintain history of user queries for tracking, analysis, and debugging purposes.

Improved Error Explanation Module – Provide user-friendly explanations for SQL errors and suggest corrections automatically.

Offline LLM Integration – Integrate local LLM models to reduce dependency on internet connectivity.

Dashboard and Data Visualization – Add graphical representations such as charts and reports for better data analysis.

Support for Complex Queries – Enhance model prompting to handle advanced joins, subqueries, and nested queries more accurately.

Performance Optimization – Implement caching mechanisms to reduce API calls and improve response time.