Google Cloud generative AI

Project Documentation format

1. Introduction

• **Project Title:** IntelliSQL: Intelligent SQL Querying with LLMs Using Gemini Pro

TEAM ID: LTVIP2026TMIDS73975

Team Members:

Name	Role
Shaik Aliya	FrontEnd
Varisetti Bhavya	BackEnd
Odugu Jahnavi	Database,Testing
Praveena Vemula	AI Integration

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 NL 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 NL 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 NLP and database management, the project demonstrates how AI can automate structured data retrieval and improve accessibility in modern data-driven applications.

Features:

• NLP to SQL conversion using Gemini Pro
• Secure relational database connectivity

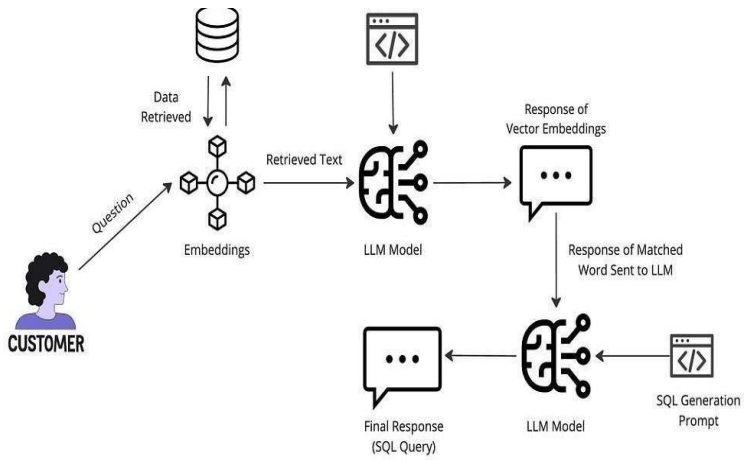
- Query validation and execution
- Data visualization (tables and charts)
- Query history management
- JWT-based authentication

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.



4. Setup Instructions

- **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

Test by entering queries

6. Cloud Deployment Streamlit Cloud – FreeTier

- 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

- **Client:** Contains React components, pages, services, and UI assets
- **Server:** Contains routes, controllers, models, middleware, and Gemini integration logic.

6. Running the Application

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

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

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

Step 4: Using the Application

1. Enter a natural language query such as:

"Show all employees with age greater than 40"

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.

7. API Documentation

- POST /api/generate-sql – Convert NLto SQL.
- POST /api/execute-query – Execute SQL and return results.

GET /api/history – Retrieve past queries

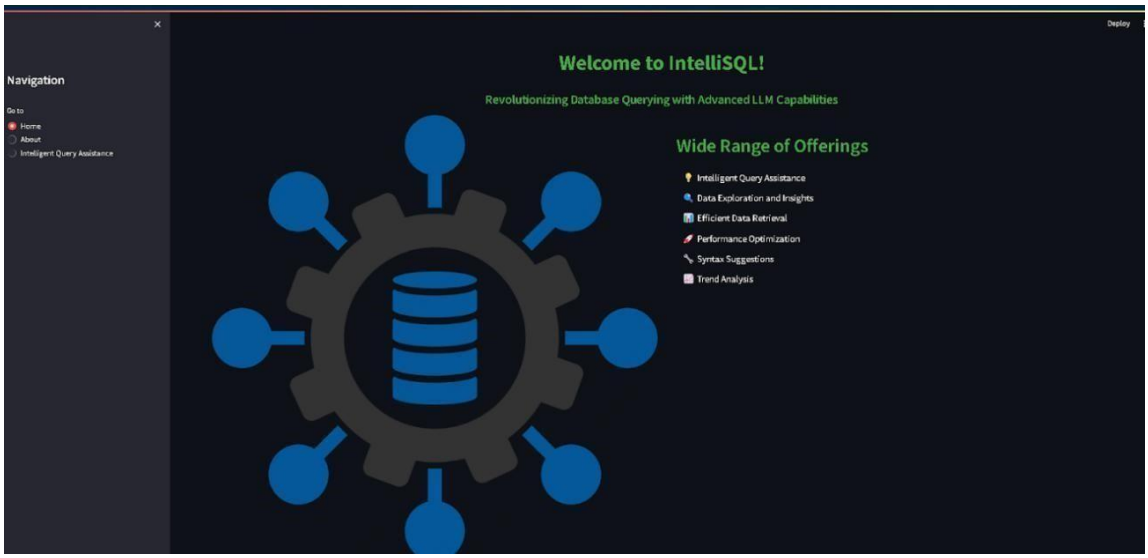
8. Authentication • JWT-based authentication ensures secure access. Role-based authorization controls database query permissions.

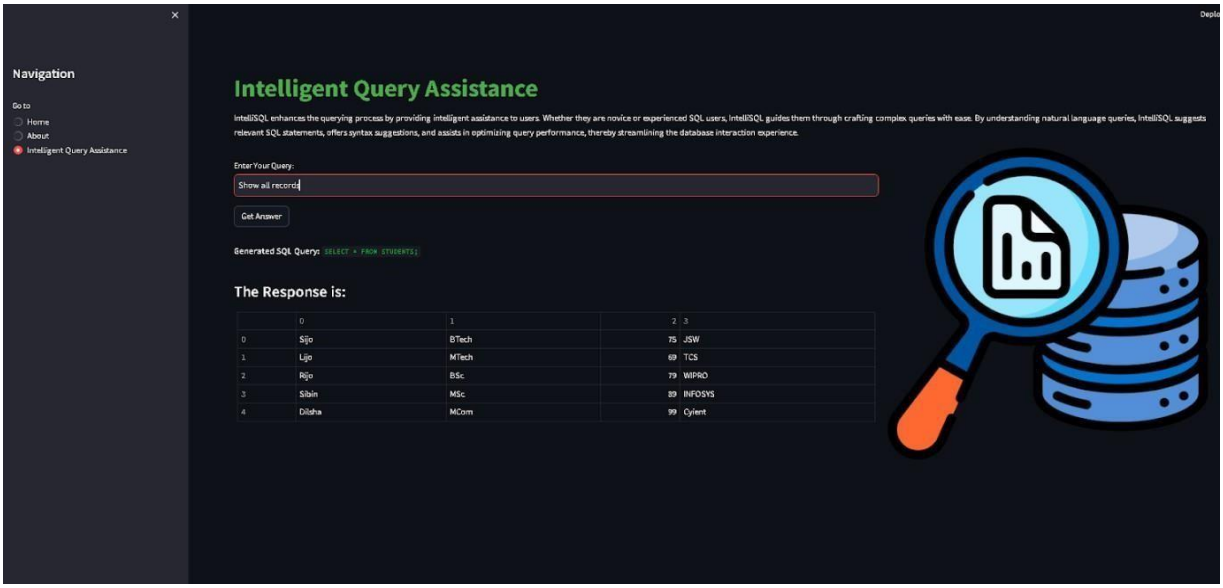
9. User Interface

• Includes login page, dashboard for query input, SQL output display, results table, and graphical visualization section.

10. Testing • Unit testing for APIs and integration testing for NL-to-SQL workflow. Performance testing for large datasets.

11. Screenshots or Demo





12. Known Issues

• SQL accuracy depends on database schema clarity.

• Complex nested queries may need refinement.

• Performance varies with dataset size

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.