

IntelliSQL: Intelligent SQL Querying with LLMs Using Gemini Pro

1. INTRODUCTION

1.1 Project Overview

IntelliSQL is an AI-powered intelligent query system that converts natural language input into structured SQL queries using Google's Gemini Pro Large Language Model (LLM). The system allows users to interact with databases without requiring knowledge of SQL syntax. Instead of writing complex SQL queries, users can simply type their questions in plain English, and the system automatically generates and executes the corresponding SQL statement.

This project integrates Natural Language Processing (NLP), Large Language Models (LLMs), and database systems to create a smart and user-friendly querying platform.

1.2 Purpose

The purpose of IntelliSQL is to bridge the gap between non-technical users and database systems. Many users struggle with writing SQL queries due to lack of technical knowledge. This project aims to:

- Simplify database interaction
- Reduce dependency on SQL expertise
- Improve productivity
- Provide intelligent query automation

2. IDEATION PHASE

2.1 Problem Statement

Many organizations store large amounts of structured data in databases. However, accessing this data requires knowledge of SQL. Non-technical users find it difficult to retrieve information efficiently. There is a need for a system that converts natural language queries into SQL automatically.

2.2 Empathy Map Canvas

- Users say: "I don't know SQL but I need data."
- Users think: "Why is it so difficult to get simple reports?" 🧠 Users feel: Frustrated with technical complexity.
- Users do: Depend on developers for small queries.

IntelliSQL addresses these pain points by providing a natural language interface.

2.3 Brainstorming

During brainstorming, we explored:

- Rule-based SQL generation
- Predefined templates
- Machine learning approaches
- Large Language Models

We selected Gemini Pro LLM because it provides advanced natural language understanding and dynamic SQL generation capability.

3. REQUIREMENT ANALYSIS

3.1 Customer Journey map

1. User logs into the application
2. User enters natural language query
3. System processes query
4. SQL is generated
5. Query executed on database
6. Results displayed

3.2 Solution Requirement

Functional Requirements

- Natural language input
- LLM-based SQL generation
- SQL validation
- Database execution
- Result display
- Error handling

Non-Functional Requirements

- Fast response time
- Secure API handling
- Reliable query generation
- Scalable architecture

3.3 Data Flow Diagram

User enters query

1. Backend sends query to Gemini Pro
2. Gemini generates SQL
3. SQL validated
4. SQL executed on database
5. Results returned to user

3.4 Technology Stack

- Frontend: Streamlit / Flask
- Backend: Python
- LLM: Gemini Pro (Google AI)
- Database: MySQL / SQLite
- Deployment: Local / Cloud

4. PROJECT DESIGN

4.1 Problem Solution Fit

The system directly solves the problem of SQL dependency by enabling natural language interaction. It improves accessibility and reduces technical barriers.

4.2 Proposed Solution

The proposed solution is a web-based application that integrates Gemini Pro API with a structured database. The application processes user queries, converts them into SQL, executes them, and displays results in real time.

4.3 Solution Architecture

The architecture consists of:

- User Interface Layer
- Application Logic Layer
- LLM Integration Layer • Database Layer

The system separates frontend, backend, and database components for modularity and scalability.

5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

1. Week 1: Requirement gathering & design
2. Week 2: Frontend development
3. Week 3: Backend & API integration
4. Week 4: Testing & optimization
5. Week 5: Documentation & deployment

6. FUNCTIONAL AND PERFORMANCE TESTING

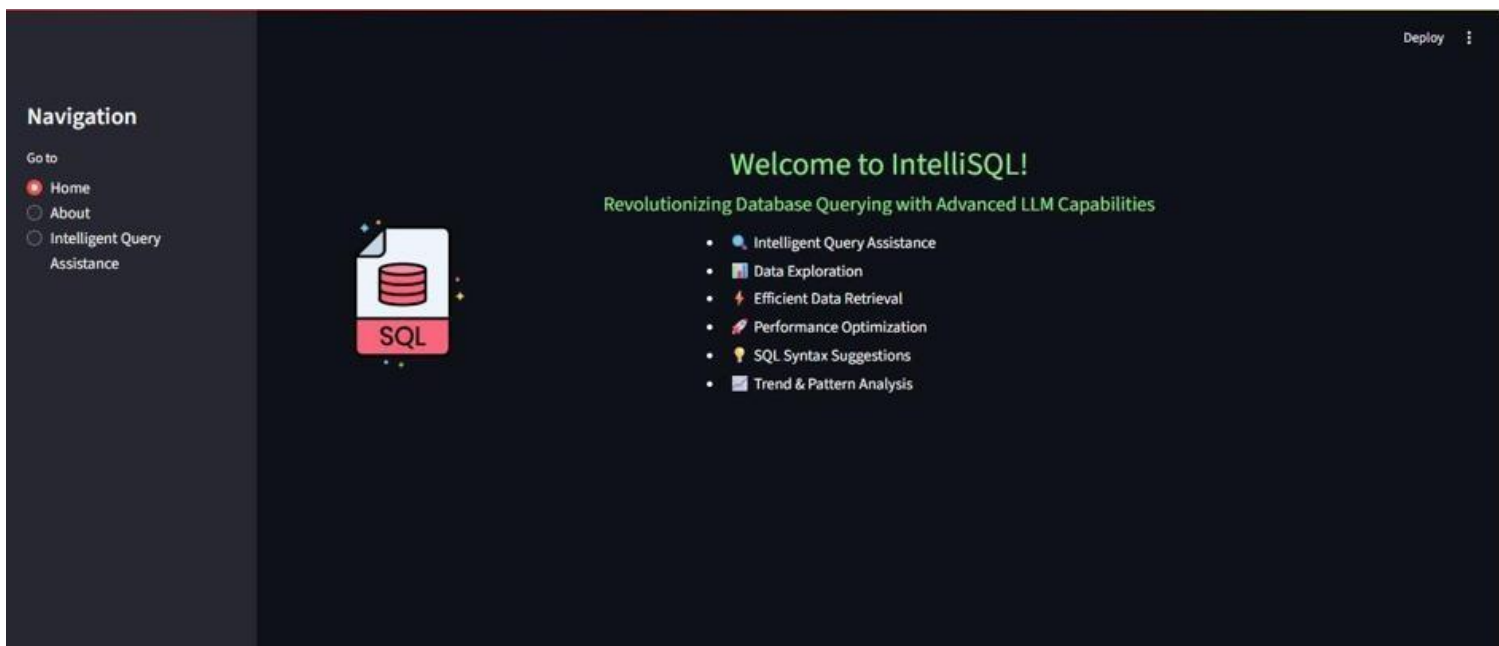
6.1 Performance Testing

- Response time under 3–5 seconds
- API reliability testing
- Multiple query handling
- Large dataset query testing

Results showed stable performance under normal load.

7. RESULTS

7.1 Output Screenshots



Navigation

Go to

- ☐ Home
- ☒ About
- ☐ Intelligent Query Assistance

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.



Navigation

Go to

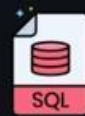
- ☐ Home
- ☐ About
- ☒ Intelligent Query Assistance

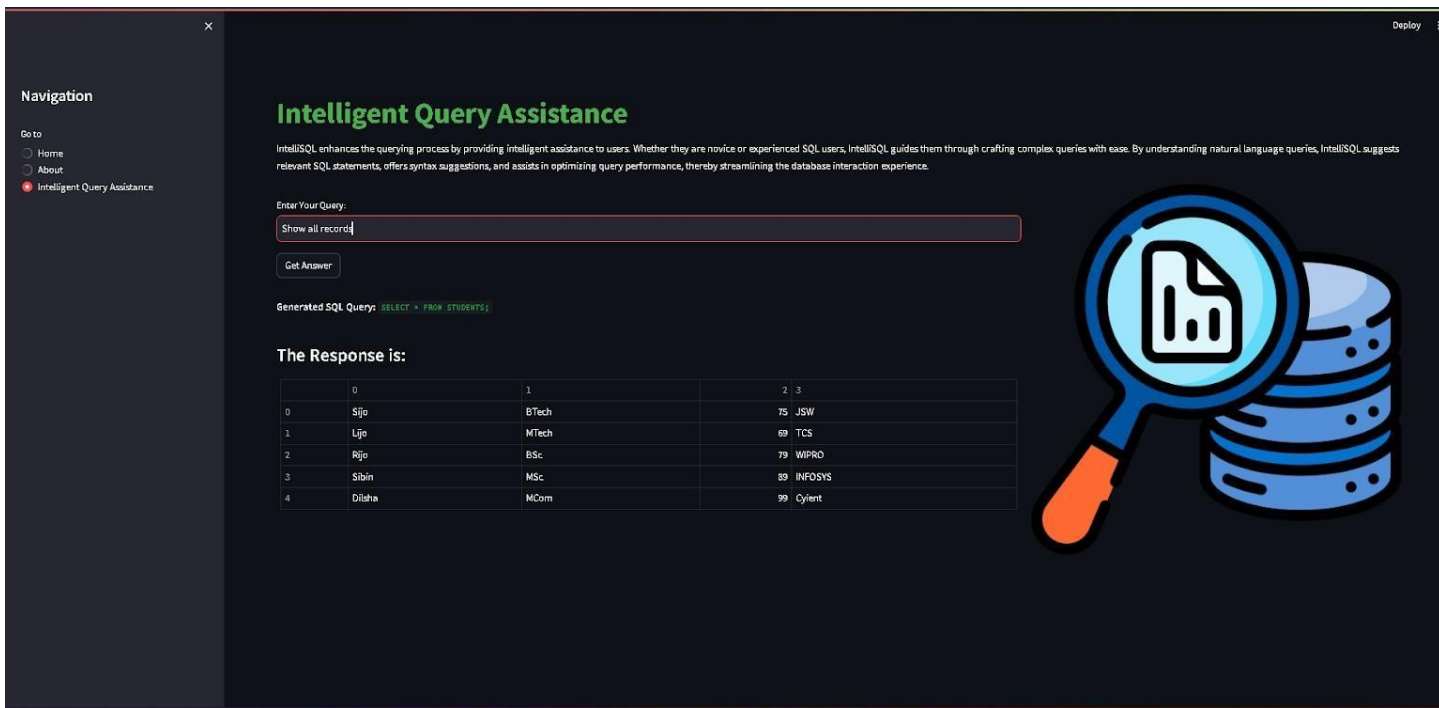
Intelligent Query Assistance

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:

Get Answer





8. ADVANTAGES & DISADVANTAGES Advantages

- No SQL knowledge required
- Time-saving
- Intelligent automation
- User-friendly interface
- Scalable solution

Disadvantages

- Depends on API availability
- Complex queries may require prompt tuning
- Internet required for LLM API

9. CONCLUSION

IntelliSQL successfully demonstrates how Large Language Models can simplify database querying. The integration of Gemini Pro enables accurate and dynamic SQL generation. The project provides an efficient and user-friendly solution for non-technical users to interact with structured databases. It showcases the power of AI-driven automation in database management systems.

10. FUTURE SCOPE

1. Support for multiple databases
2. Voice-based query input
3. Advanced query optimization

4. Role-based access control
5. Query history tracking
6. Multi-language support

11. APPENDIX

GitHub: <https://github.com/HANIEF1103/IntelliSQL-SQL-Querying-with-LLMs-Using-Gemini-Pro.git>

Demo Video: https://drive.google.com/file/d/12s_Ud8trYbjfO4cdPK4BLejOggwWCEZQ/view?usp=drivesdk