AI-powered Customer Service for Telecom

1. Introduction

Our project is an AI-powered Phone Plan Assistant designed to simplify user access to mobile subscription plans through a natural language interface. By integrating OpenAI's language model with a structured SQL database, the assistant enables users to interact conversationally to retrieve, compare, or switch plans without needing technical knowledge.

1.1 Motivation

Mobile subscription plans can be complicated, and many users find it frustrating to contact customer service or visit a store just to get basic information. We aim to use AI to make this process faster and more accessible. With advances in large language models (LLMs) such as OpenAI's GPT, users can now ask questions in natural language and receive accurate, real-time answers, improving the overall customer experience.

1.2 Innovation

Unlike traditional rule-based chatbots (e.g. IBM Watson), our system generates SQL queries dynamically from open-ended user input. This allows for more personalized and flexible responses. It is designed with a user-centric approach—users don't need prior knowledge about mobile plans or SQL. They can simply ask the assistant questions like "Which plans offer unlimited data?" or "Can I downgrade to a student plan?". Additionally, our modular architecture and secure SQL verification process enable easy integration with existing telecom platforms.

1.3 Repo

Link: https://github.com/yfl423/LLM-based-Agent-Service

2. Literature Review

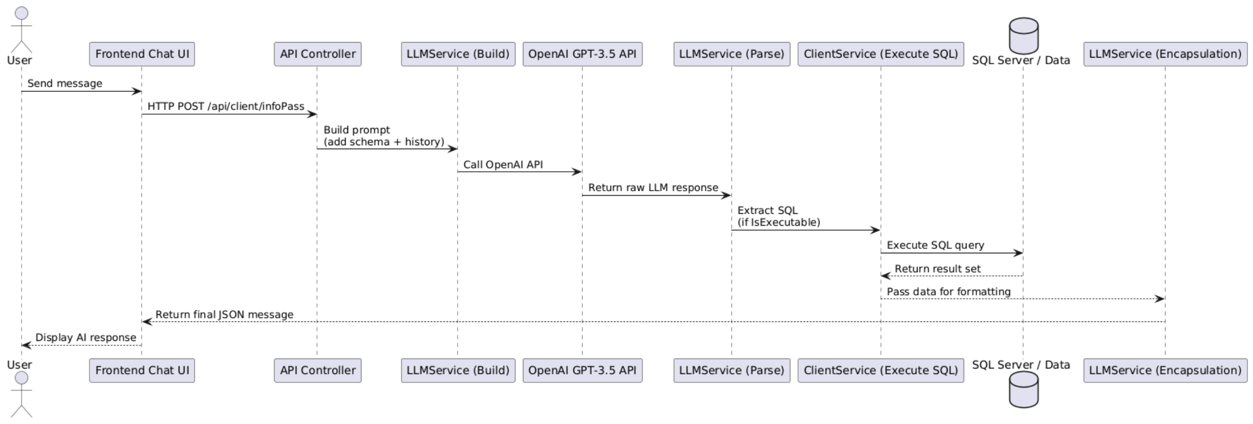
Our project utilizes the OpenAI API to interpret users' natural language input.Then the system will dynamically generate SQL commands based on their intent. This integration enables real-time understanding and execution of questions such as "What phone plans include high data?" or "Can I switch to the Youth Plan?"
Trummer [1] introduced the CodexDB system that uses GPT-based models to translate human language into SQL. Their work demonstrates the feasibility of using LLMs in complex database environments. Unlike CodexDB, our AI assistant incorporates real-time prompt engineering and schema validation to ensure safe execution.
Mohammadjafari, Maida, andGottumukkala [2] conducted a comprehensive review of text-to-SQL systems using LLMs. This highlights the critical role of prompt engineering.

Similarly, our project also heavily relies on prompts to guide the model in producing accurate and valid SQL outputs.
Li and others [3] analyzed real-world readiness of text-to-SQL systems, highlighting issues of reliability and safety. In line with their findings, our system performs rigorous review of GPT-generated SQL to eliminate threats such as unintended table deletion or data leakage.
Zhang and Zhou [4] discussed text-to-SQL systems that can be attacked using prompt-based SQL injection. Their work highlights the critical importance of inspecting generated queries and prompts for hidden triggers and payloads. Inspired by their findings, our backend pipeline includes safety checks for risky patterns like DROP TABLE to prevent malicious injections via prompt manipulation.

3. Methodology

The AI Phone Plan Assistant is implemented as a full-stack web application that integrates frontend interaction and backend logic. The system uses OpenAI's language model to dynamically translate user input into structured database queries. Our project follows a modular pipeline consisting of frontend UI, natural language processing, prompt engineering, SQL generation, and backend validation and execution.

3.1 End-to-End Workflow



3.2 Frontend Interaction

The frontend is built using HTML, CSS, and JavaScript. It presents a chat-based user interface where users can type their questions in natural language. Every user message is captured by JavaScript and sent to the backend server using a fetch() API call. The interface supports real-time message display, user input tracking, and auto-scroll for user experience.

3.3 Backend Architecture

- Backend: ASP.NET Core Web API (.NET 7)
- LLM Integration: OpenAI GPT-3.5 Turbo
- Database: SQL Server in Docker

- Session Cache: ASP.NET MemoryCache

3.4 Data Schema Design

The system contains two main tables: one is plan_table, which stores all the mobile phone plans and its properties. The other is client_table, which stores customers information like name, phone number and so on.
This schema is designed to support both quer general plan details and switch personalized plan after user authentication. The system enables secure and contextual data retrieval through linking users to their subscribed plans.

3.5 Core Features

3.5.1 Query-Based Plan Discovery

Users can ask human language questions like “What’s the cheapest unlimited plan?”. The system interprets the question using the OpenAI GPT model. And then generates an SQL query to retrieve relevant results from the database.

3.5.2 Plan Comparison & Follow-Up

The system uses GPT to generate natural language responses that include details such as plan name, price, data allowance, and other details. These responses are dynamically generated based on SQL query results. Users can compare multiple plans based on attributes like data, price, or voice minutes. The chatbot also supports follow-up questions, allowing users to refine their queries in a conversational and interactive manner.

3.5.3 Secure Client Lookup (Authentication)

For personal queries like “What is my plan?”. The system requests user authentication, including name, phone number, and date of birth. This ensures that no unauthorized user can access another person’s data.

3.5.4 SQL Injection Protection

The system actively detects and blocks harmful prompts such as “DROP TABLE client_table”. It returns a warning message like “Sorry, this request seems unsafe,” to prevent dangerous SQL execution.

3.5.5 Plan Modification Support

Users can type requests like “I want to switch to Basic Plan.” The system verifies whether the plan exists, updates the user record, and confirms the change accordingly.

4. Testing & Evaluation

To ensure the AI Phone Plan Assistant performs reliably across different use cases, a series of structured tests were conducted covering both functional and security dimensions. Testing was carried out manually through the frontend interface as well as through Swagger UI for backend validation.

4.1 Functional Testing

We tested the system using various natural language prompts. Key scenarios included:

- Plans query: Users asked open-ended questions such as “What phone plans are available?” or “What is the cheapest plan with high data?” The system correctly returned filtered plans with relevant details.
- Specific plan check: Prompts like “I want to know more about Unlimited Everything” successfully returned full plan information.
- Follow-up interactions: After receiving one result, users asked follow-up questions like “Does it include SMS?” and the system maintained context and responded appropriately.
- Plan switch: Users’ input such as “Switch to Youth Plan” triggered a successful update of the user’s plan.

4.2 Security Testing

We evaluated how well the system handles potentially malicious input:

- SQL Injection Protection: Queries like DROP TABLE client_table were blocked, and the system returned a message indicating unsafe request.
- Authentication Verification: Before accessing personal information, the system asks for name, phone number, and date of birth, ensuring only verified users can access client records.
- Data Filtering: Even after GPT generates SQL, a backend layer checks for dangerous instructions before executing.

4.3 User Testing

We had 5 real users test the system. Each user performed a series of queries covering plan search, comparison, and switching. Feedback showed:

- The assistant was easy to use and intuitive.
- Follow-up questions made the interaction feel conversational.
- Users felt their data was protected through verification prompts.
- LLM could forget initial requirements as conversation goes deeper, such as the response format is not as expected.
- The always-true conditions such as OR 1=1 can successfully attract the system.
- Test users information:
    - ❖ Yaning Li 25 Male
    - ❖ Ze Wang 34 Male
    - ❖ Cornelia Chu 29 Female

- ❖ Yuxiao Wang 24 Female
- ❖ Benchi Zhang 30 Male

5. Lessons Learned & Future Improvements

5.1 Lessons Learned

Throughout the development of the AI Phone Plan Assistant, we encountered several practical challenges and solutions:

- As conversations progress, the LLM could forget initial context or output formats (e.g., JSON/T-SQL) may become inconsistent. To address this, we implemented try-catch error handling and added logic to check response conformity.

- To avoid SQL injection vulnerabilities, we used special characters like backticks to highlight user inputs. This helped reduce risks when injecting inputs into prompt templates.

- Prompt structure significantly affects GPT output. Injecting schema rules and using structured prompt templates stabilized response generation.

5.2 Future Improvements

We received valuable feedback from real person testers during our demo and identified potential enhancements:

- Integrate a vector-store–backed retrieval step so the LLM can fetch up-to-date product docs, regulatory rules, or FAQ knowledge before generating SQL or explanations. This reduces hallucinations and keeps recommendations accurate.

- Fine-tune a smaller LLM on historical telecom customer dialogs, plan catalogs, and billing scripts to improve SQL-generation accuracy and naturalness of plan-change conversations.

- Add voice input, mobile optimization, and filterable plan views to improve usability.

6. Contributions & Self-scoring

6.1 Fenglong Yang

6.1.1 Contributions

- Engineered a robust backend architecture and developed scalable APIs leveraging .NET 7, ensuring high performance and maintainability across services.

- Crafted and optimized prompt strategies with OpenAI to intelligently interpret user input, translate it into structured T-SQL queries, and execute seamless database interactions.
- Led system-level debugging and resilience enhancements, proactively diagnosing and mitigating exceptions and edge cases triggered by LLM-driven workflows.

6.1.2 Self-scoring table

Evaluation Category	Description	Self-Score (out of 10)
Innovation and Creativity	Integrated OpenAI GPT with secure SQL logic and natural conversation design	9
Technical Complexity	Implemented full-stack structure, OpenAI parsing, and error handling	8
Core Functionality Implementation	Designed chatbot UI, backend routes, and database linking	10
Testing & Debugging	Conducted multi-scenario testing and handled edge case logic	10
Documentation	Wrote detailed report and organized component structure	8
UI/UX Design	Built a functional, user-friendly front end	9
Security Handling	Implemented SQL injection prevention and ID verification logic	8
Code Quality & Modularity	Maintained clean, readable, and modularized code	9
Communication & Demo	Shared updates, ran test cases with users, received feedback	8
Overall Execution	Project completed fully with working back-end and front-end integration	10

6.2 Changhui Sun

6.2.1 Contributions

- Designed and developed a chatbot user interface that allows customers to interact naturally with the AI assistant.

- Integrated frontend and backend systems to enable real-time communication and response handling.

- Applied prompt engineering techniques to wrap SQL query results and generate clear, human-readable responses.

- Designed and implemented the backend database schema, including tables for phone plans and client information.

6.2.2 Self-scoring table

Evaluation Category	Description	Self-Score (out of 10)
Innovation and Creativity	Integrated OpenAI GPT with secure SQL logic and natural conversation design	8
Technical Complexity	Implemented full-stack structure, OpenAI parsing, and error handling	7
Core Functionality Implementation	Designed chatbot UI, backend routes, and database linking	10
Testing & Debugging	Conducted multi-scenario testing and handled edge case logic	8
Documentation	Wrote detailed report and organized component structure	9
UI/UX Design	Built a functional, user-friendly front end	9
Security Handling	Implemented SQL injection prevention and ID verification logic	7
Code Quality & Modularity	Maintained clean, readable, and modularized code	9

Communication & Demo	Shared updates, ran test cases with users, received feedback	9
Overall Execution	Project completed fully with working back-end and front-end integration	9

6. Conclusion

In this project, we designed and implemented an AI-powered Phone Plan Assistant that enables users to interact with AI through natural language. By integrating OpenAI's GPT model with SQL backend processing, we successfully demonstrated how LLMs can bridge the gap between everyday user intent and structured database queries.
The system supports intelligent features such as plan discovery, comparison, secure user lookup, and SQL injection protection. It dynamically generates SQL commands based on user queries and safely executes them to return real-time information. This showcases a practical and secure way to use LLMs in database interaction without requiring prior technical knowledge from the user.
Our project highlights how AI can simplify customer service workflows, increase accessibility, and ensure security through prompt engineering and backend filtering. This framework can be extended to various industries beyond telecom, such as banking, healthcare, or retail.
We believe that with continued refinement—like integrating domain-specific knowledge and retrieval-augmented generation—the assistant can evolve into a powerful enterprise solution for real-time, natural language database access.

7. References

[1] I. Trummer, "CodexDB: Integrating LLMs with SQL Databases for Natural Language Querying," Advances in Neural Information Processing Systems (NeurIPS), 2022.
[2] M. Mohammadjafari, A. Maida, and R. Gottumukkala, "A Review of Text-to-SQL Systems with Large Language Models," arXiv preprint, arXiv:2305.17875, 2023.
[3] L. Li, Y. Yao, W. Li, and Y. Zhang, "Are We Ready for Production? A Case Study of NL-to-SQL Systems in Practice," Proc. VLDB Endowment, vol. 15, no. 12, pp. 3310–3322, 2022.
[4] Z. Zhang and M. Zhou, "Prompt-based SQL Injection Attacks Against Text-to-SQL Systems," arXiv preprint, arXiv:2306.16429, 2023.