Database Dashers

Members:

Bibhudatta Bhanja 21CS10015

Dhruv Sarkar 21CS30063

Kaushal Kumar Singh 21CS30029

Snigdha Biswas 21CS30049

Anuron Chakraborty 21CS10085

Ditch the Search, Chat Your Way to Your Dream Car with CAR GURU!

Tired of endless scrolling and confusing filters? CAR GURU revolutionizes your used car search with the power of natural language.

Just tell our friendly AI assistant what you're looking for - a fuel-efficient sedan, a spacious family SUV, or a sporty coupe with a sunroof. CAR GURU understands your needs and uses its knowledge to search a vast database of used cars, finding the perfect match for you.

No more deciphering car jargon! Ask questions in plain English and get clear, informative answers.

Objective:

The objective of the NaturalQuery project is to explore and implement the integration of Large Language Models (LLMs) with SQL databases. The aim is to leverage the capabilities of LLMs to enable users to query databases using natural language, thereby simplifying data retrieval and analysis processes. By bridging the gap between traditional SQL queries and human language, the project seeks to democratize access to structured data and empower users with varying levels of technical expertise to harness the insights stored within databases.

Methodology:

The project methodology encompasses several key steps, including data collection and preprocessing, model selection and training, evaluation metrics and techniques, and integration with PostgreSQL databases. A dataset of natural language questions paired with corresponding SQL queries is curated and preprocessed to prepare it for model training. An open-source LLM architecture, such as Gemini, GPT-3.5 or BERT, is selected and fine-tuned on the dataset to learn the mapping from natural language to SQL queries. The trained model is evaluated using performance metrics such as accuracy, precision, and recall, and integrated with PostgreSQL databases to enable natural language querying.

- **RAG (Retrieval-Augmented Generation):** This innovative technique enhances LLMs by providing them with relevant information retrieved from external sources before generating text. Utilizing libraries such as LangChain and Hugging Face, RAG ensures that responses are grounded in up-to-date facts, thereby improving the reliability and informativeness of LLM-generated content.

- **Gemini (Generative Minimalistic InteractNlp Interface):** Gemini is a minimalist yet powerful tool designed to facilitate natural language interaction with databases. Unlike traditional interfaces, Gemini simplifies the process of querying databases by allowing users to pose questions in plain language without the need for prior knowledge of complex query languages.

- In the context of NaturalQuery, Gemini serves as the foundation for enabling seamless communication between users and SQL databases. By leveraging its intuitive interface and natural language processing capabilities, Gemini empowers users to express their queries in everyday language, simplifying the data retrieval process and enhancing accessibility to structured data.

- **PostgreSQL:** PostgreSQL, often referred to as Postgres, is selected as the database management system for this project. It is a free, open-source database system renowned for its reliability, security, and scalability. Additionally, PostgreSQL supports the industry-standard SQL language, making it accessible to a wide range of users. Its robust feature set and extensive community support make it an ideal choice for storing and managing the structured data required for natural language querying.

- **Streamlit:** Streamlit streamlines the creation of data applications for data scientists and machine learning engineers. It eliminates the need for time-consuming web development, allowing construction of attractive and interactive data apps using simple Python code. Imagine having a powerful machine learning model but facing challenges in sharing insights with colleagues. Streamlit empowers you to rapidly create user-friendly apps where anyone can interact with your model, visualize results, and adjust parameters – all without requiring web development expertise. Streamlit shines in rapid prototyping, facilitating experimentation with data exploration and visualization tools without extensive coding. It simplifies data sharing by enabling effortless dissemination of insights with collaborators through user-friendly web apps. Furthermore, Streamlit excels in machine learning deployment, allowing you to deploy your models as interactive applications, empowering users to explore predictions. By bridging the gap between data and actionable insights, Streamlit positions itself as the perfect tool to showcase your work and democratize access to powerful data analysis.

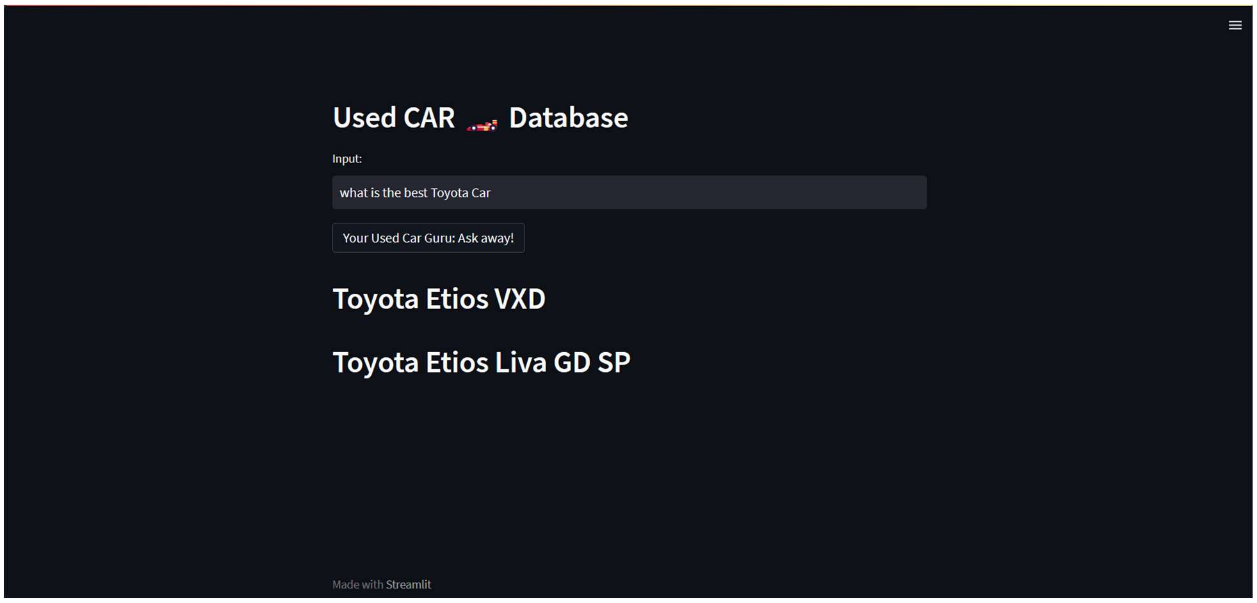
Data Collection and Preprocessing: The initial phase of the methodology involves the careful curation and preprocessing of a dataset. We used a Indian used cars dataset which was available on Kaggle. Lot of filtering and cleaning techniques were used in order to clean the dataset, and finally 7 attributes were selected for the representation: name, selling price, km driven, mileage, seats, petrol and transmission

Model Selection: Once the dataset is prepared, the next phase focuses on selecting an appropriate LLM architecture for the natural language to SQL query generation task. Popular LLM architectures such as Gemini, GPT-3.5, BERT, and T5 are considered, each offering unique advantages and capabilities. After meticulous testing we finalized the Gemini-Pro model as our LLM, due to its indiscrete access, and faster and accurate response time in comparision to the other models in fray. The chosen model was not trained again, as using this pipeline created the text to SQL conversion is seamless,, and doesn't require any particular computation to

Results, Evaluation Metrics and Techniques: For our project we are basically comparing generated results with the actual output of SQL schema, and more often then not the query results are successful. Our Framework works well for all sorts of queries, the results for the same are displayed below:

Query: what is the best Toyota Car

Results:



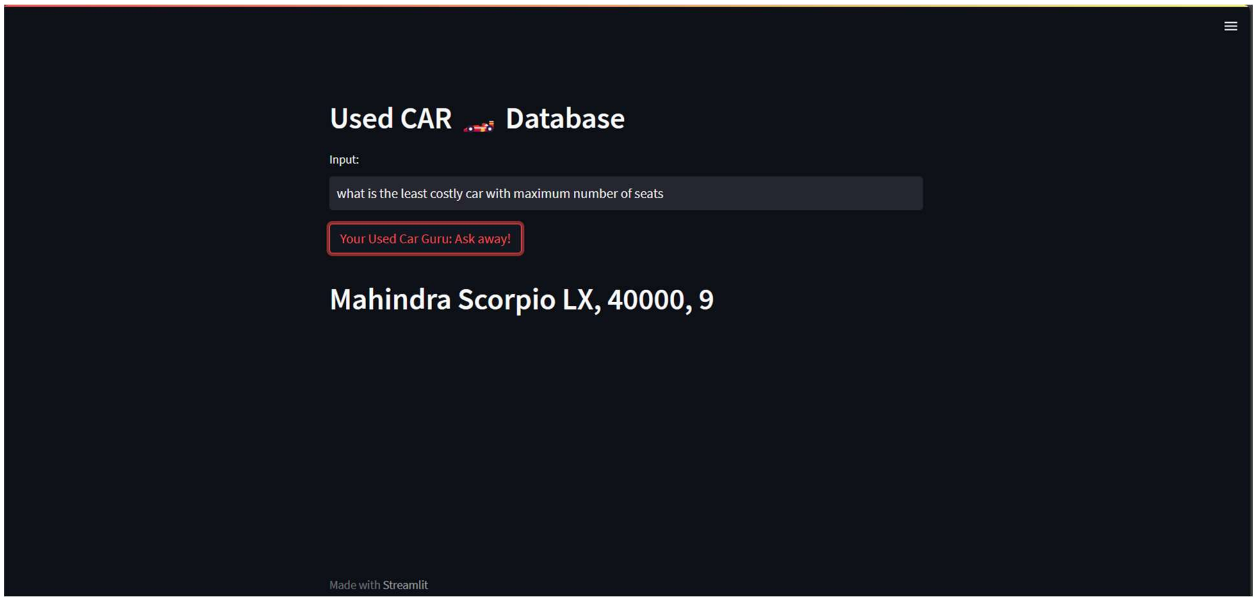
what is the best Toyota Car

Toyota Etios VXD

Toyota Etios Liva GD SP

Query: what is the least costly car with maximum number of seats

Results:

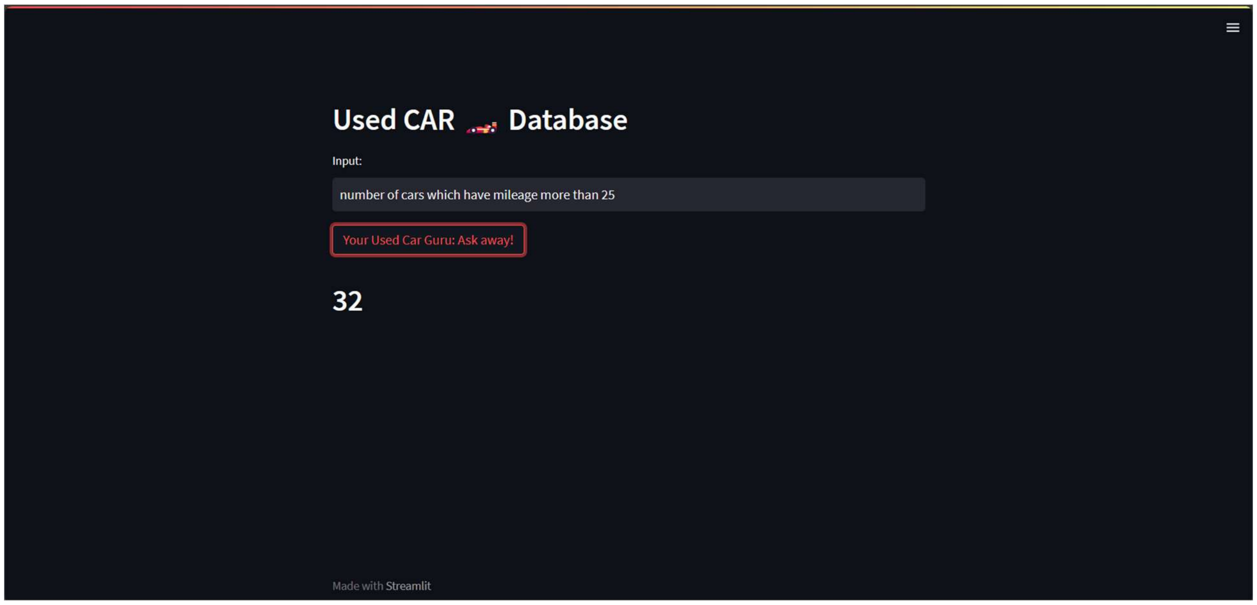


what is the least costly car with maximum number of seats

Mahindra Scorpio LX, 40000, 9

Query: number of cars which have mileage more than 25

Results:


Used CAR Database

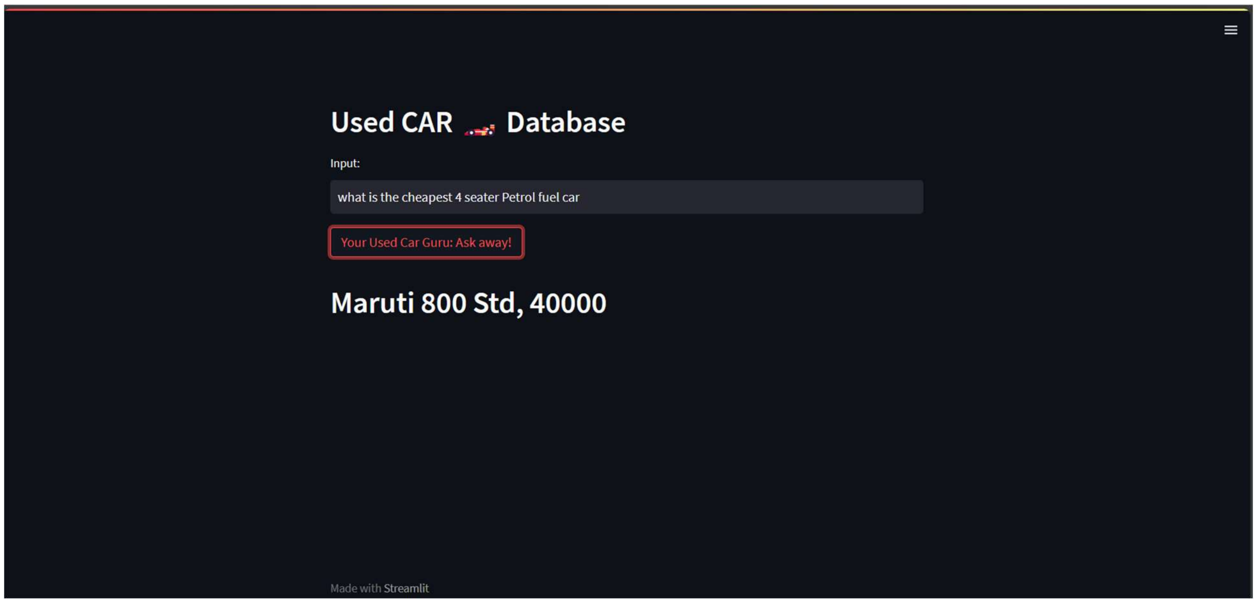
Input:

number of cars which have mileage more than 25

32

Query: what is the cheapest 4 seater petrol fuel car

Results:


Used CAR Database

Input:

what is the cheapest 4 seater Petrol fuel car

Maruti 800 Std, 40000

Integration with PostgreSQL Database: The final phase of the methodology involves integrating the trained LLM model with a PostgreSQL database to enable natural language querying. An interface is developed to accept user queries in natural language format and pass them to the LLM model for translation into SQL queries. The generated SQL queries are then executed on the PostgreSQL database, and the retrieved results are presented to the user in a human-readable format. The integration process involves implementing robust error handling mechanisms to ensure the system's reliability and scalability in real-world applications.

Code for the Project: https://github.com/bbhanja1809/CAR-GURU.git

Obstacles Faced:

During the usage of Postgresql for storing the database there were several latency issues that were faced by us due to which we had to switch to sqlite which is a SQL implementation in python. Apart from this, while using LLM models like GPT 3.5, the results were optimal and latency was less, but due to a credit limit enforced by OpenAI, we could process only few queries with it before finally switching over to Google's Gemini-Pro.

Future Optimizations:

1. Enhanced Retrieval Strategies:

- **Deeper Contextual Understanding:** Utilize more sophisticated retrieval methods like dense retrieval models to improve the matching between user prompts and relevant database descriptions or past queries. This can help Gemini Pro capture the nuances of user intent and retrieve the most appropriate information for query generation.
- **Domain-Specific Knowledge Graphs:** Integrate domain-specific knowledge graphs into the retrieval process. These knowledge graphs can act as a structured source of information about the database schema and relationships between entities, enabling Gemini Pro to understand the underlying data structure and generate more accurate queries.

2. Advanced Prompt Engineering Techniques:

- **Adaptive Prompting:** Develop techniques that dynamically adjust the prompt based on the complexity of the user's request. For simpler queries, a concise prompt might suffice. For complex queries, a more elaborate prompt with additional instructions or examples could guide Gemini Pro towards generating a more accurate SQL query.
- **Active Learning:** Implement active learning techniques where Gemini Pro iteratively refines its prompts based on user feedback and the success of previous queries. This allows the system to learn from interactions and continuously improve its ability to translate natural language prompts into correct SQL queries.

3. Robust Error Handling and Explainability:

- **Improved Error Detection and Correction:** Develop robust mechanisms for detecting and correcting errors in the generated SQL queries. This could involve syntax checks, data type validation, and consistency checks with the database schema.
- **Explainable AI for Query Generation:** Integrate explainability methods into Gemini Pro. This would allow users to understand the reasoning behind the generated SQL query, fostering trust and enabling users to identify potential issues or areas for improvement.

References:

- **Streamlit:**
  - Streamlit Documentation: https://docs.streamlit.io/
  - Streamlit Community Tutorial: https://docs.streamlit.io/develop/tutorials
- **Langchain:**
  - Langchain Documentation: https://github.com/langchain-ai
  - Langchain Blog: https://medium.com/@jh.baek.sd/creating-an-ai-web-service-using-langchain-with-streamlit-using-llama-2-or-chatgpt4-83a824e19435
- **Llama Index:**
  - Llama Index GitHub Repository: https://github.com/run-llama/llama_index
  - Blog post using Llama Index: https://github.com/facebookresearch/llama-recipes/issues/253
- **Retrieval-Augmented Generation (RAG):**
  - A Gentle Introduction to the Retrieval-Augmented Generation Paradigm: https://huggingface.co/docs/transformers/en/model_doc/rag
  - Blog post on RAG implementation: https://medium.com/@abdulll8392/chat-with-your-documents-retrieval-augmented-generation-using-llama-index-3d264fddbcb5
- **Large Language Model (LLM):**
  - The Illustrated GPT-3: https://openai.com/product
  - Blog post on understanding LLMs: https://www.theverge.com/2022/11/2/23434360/google-1000-languages-initiative-ai-llm-research-project