SUPERIOR UNIVERSITY

Programming for Artificial Intelligence (lab)

Project Documentation

Name:

Ali Maqsood.

Roll no:

SU92-BSAIM-F23-050.

Department:

Software Engineering Department.

Program:

Artificial Intelligence.

Section:

BSAI-4A

Automotive Fault Diagnosis Assistant

Project Overview:

This project is a web-based tool designed to help car owners and mechanics diagnose vehicle problems. It allows users to describe the symptoms of their car in plain language and receive step-by-step guidance on what could be wrong and what checks to perform.

The system combines Natural Language Processing (NLP) and Generative AI (GenAI) to understand the user's input and generate accurate, easy-to-follow responses.

Project Category:

This project falls into the NLP and Generative AI domain:

- ***NLP:*** To understand and process the user's input and find relevant past cases using sentence embeddings.
- ***Generative AI:*** To create a human-readable diagnostic explanation and step-by-step instructions using Google's Gemini AI model.

Key Features:

1. User-Friendly Interface:
    - Users can type in their car issues in plain English.
    - A clean web interface guides the user through submitting their query.
2. Semantic Understanding with NLP:
    - The system uses sentence embeddings to understand the meaning of the user's input.
    - It searches a database of known automotive issues to find the most similar cases.
3. Step-by-Step Diagnostic:
    - After finding relevant cases, the system uses Generative AI to generate clear recommendations.
    - The response includes the most likely cause and step-by-step checks the user can perform.

4. Fallback for Unknown Queries:
    - If the user enters a problem that is not in the database, the system advises consulting a professional mechanic.
5. Query History Caching:
    - The system caches responses for previously asked questions.
    - When a user submits the same query again, the cached response is returned instantly without calling the Gemini API.
    - Reduces API calls, improves response time, and saves costs.
6. Accessibility Features:
    - The results page is keyboard-friendly.
    - Users can navigate easily and read the response clearly.

Project Architecture:

1. Frontend:
    - Allows users to submit queries and displays the AI-generated diagnosis.
    - Loader animation shows while the system processes the input.
2. Backend:
    - Handles form submissions.
    - Checks query history cache before processing new requests.
    - If query exists in cache, returns the stored response immediately.
    - If query is new, passes it to the NLP module to find similar cases.
    - Sends the retrieved information to Generative AI for a human-readable answer.
    - Saves the response to history cache for future use.
    - Returns the final result to the frontend.
3. NLP Module:
    - Uses sentence-transformers to convert text into embeddings.
    - Uses FAISS for fast similarity search among past automotive cases.

4. Generative AI Module:
    - Uses Google Gemini API to produce detailed and natural language diagnostic responses.
    - Ensures output is easy to understand, formatted in step-by-step instructions.
5. History Module:
    - Manages prompt-response caching using JSON file storage.
    - Normalizes user queries (lowercasing, removing punctuation) for consistent matching.
    - Provides fast lookup of cached responses to avoid redundant API calls.
    - Persists history across application restarts.
6. Data:
    - Contains a dataset of automotive faults, symptoms, and diagnosis steps.
    - Preprocessed into “chunks” suitable for vector search and AI generation.

Technologies Used:

- *Backend:* Python, Flask
- *NLP & Vector Search:* pandas, numpy, sentence-transformers, FAISS
- *Generative AI:* Google Gemini API
- *Frontend:* HTML, CSS, JavaScript
- *Environment Management:* python-dotenv for API keys
- *Data:* CSV files with automotive faults and diagnostic steps

Workflow:

1. User types a description of the car problem.
2. The system checks if this query exists in the history cache.
3. If found in cache, the stored response is returned instantly.
4. If not in cache, the system cleans and embeds the input.
5. FAISS retrieves the top 3 most similar automotive cases.

6. These cases are sent along with the user query to Gemini AI.
7. Gemini AI generates a human-readable, step-by-step diagnostic guide.
8. The response is saved to history cache for future queries.
9. Result is displayed in the browser with a clear, accessible format.

Challenges and Solutions:

- Handling unknown queries:
  - Implemented a fallback message advising professional help.
- Ensuring reliable AI output:
  - Used structured prompts and verified Gemini API responses.
- Fast and accurate retrieval:
  - Pre-processed data and embeddings, used FAISS for efficient similarity search.
- User experience:
  - Added loader animations and removed unnecessary focus borders for better UI.
- Avoiding redundant API calls:
  - Implemented a query history caching system that stores prompt-response pairs.
  - Normalizes queries to handle variations (e.g., extra spaces, different cases).
  - Uses persistent JSON file storage so history survives application restarts.

Conclusion:

The Automotive Fault Diagnosis Assistant successfully combines NLP and Generative AI to provide users with understandable, actionable insights about vehicle problems. It demonstrates how AI can help automate diagnostic processes while remaining accessible and easy to use.