CAPSTONE PROJECT

Voice Assistant App

- **PRESENTED BY: Hrishi Jayesh Mehta**
- **STUDENT NAME: Hrishi Jayesh Mehta**
- **COLLEGE NAME: University Of Mumbai**
- **DEPARTMENT: Information Techonology**
- **EMAIL ID: mehtahrishi45@gmail.com**
- **AICTE STUDENT ID: STU677477443fe471735685956**

Index

Problem Statement

The increasing demand for hands-free interaction and readily available information necessitates intuitive and accessible voice-based interfaces. Existing solutions often require specific platforms or complex setups. There is a need for a modern, web-based voice assistant that offers seamless interaction through speech, provides quick responses for common queries, and can leverage AI for more complex requests, all within a user-friendly and visually appealing interface.

Proposed Statement

A modern voice assistant web application is proposed, utilizing a combination of rule-based logic for efficient handling of simple queries and a fallback AI system for addressing more complex user requests. The application will feature a user-friendly web interface with speech recognition and synthesis capabilities, creating a natural and intuitive interaction experience similar to dedicated voice assistants.

System Development Approach (Technology Used)

The system will be developed using a modular approach, separating the backend logic from the frontend presentation. The following technologies will be employed:

- **Backend:** Flask (Python) will be used to build the server-side logic, handle API requests, and manage the rule-based and AI-powered responses.
- **Frontend:** HTML, CSS, and JavaScript will be used to create the interactive user interface, including elements for voice input, text input, and displaying responses.
- **Voice Recognition & Synthesis:** The Web Speech API will be integrated into the frontend to enable speech-to-text (for user input) and text-to-speech (for assistant output) functionalities.
- **AI Integration:** The OpenRouter API will be utilized as a fallback mechanism to process user queries that do not match the predefined rule-based responses, providing more comprehensive and context-aware answers.
- **User Interface and Experience:** CSS animations will be implemented to enhance the user experience with smooth transitions and interactive elements.

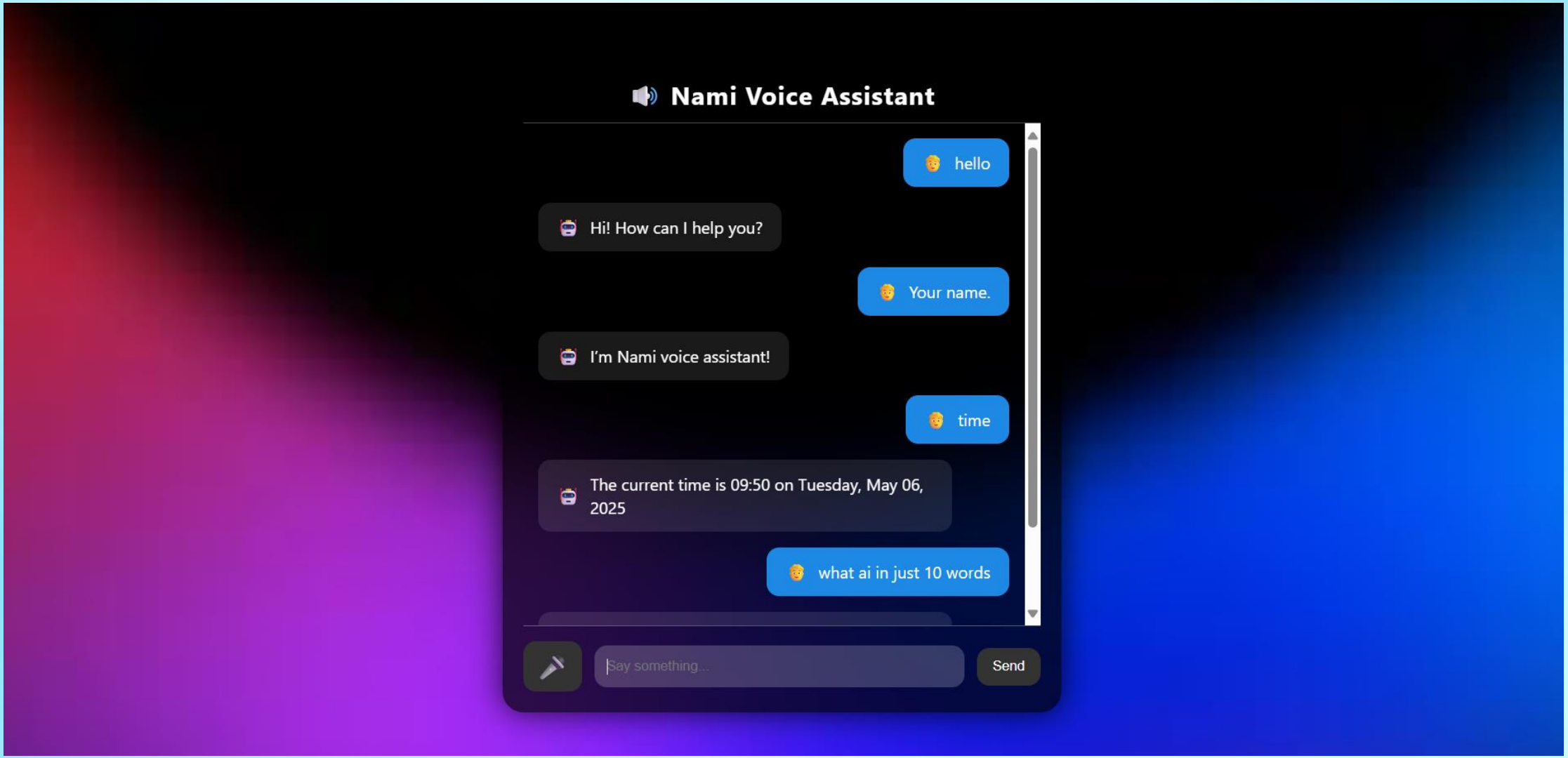
Algorithms

- **User Input:** The user provides input either through voice (captured via Web Speech API and converted to text) or by typing into the text input field.
- **Input Processing (Backend):** The frontend sends the user's text input to the Flask backend via an API request.
- **Rule-Based Matching:** The backend first checks if the user's input matches any predefined rules (e.g., "Hello," "What time is it?").
- **Rule-Based Response:** If a match is found, the backend generates the corresponding predefined response.
- **AI Fallback:** If no rule-based match is found, the backend sends the user's input to the OpenRouter API.
- **AI Response:** OpenRouter processes the input and returns an AI-generated response to the backend.
- **Response Transmission:** The backend sends the response (either rule-based or AI-generated) back to the frontend.
- **Output Display:** The frontend displays the assistant's response in the message box.
- **Speech Synthesis:** The frontend uses the Web Speech API to synthesize the assistant's text response into spoken audio.

Deployment

The application can be deployed locally by following the installation instructions provided, which involve cloning the repository, setting up a virtual environment, installing dependencies, configuring the OpenRouter API key, and running the Flask application. This will host the web app on a local development server. For wider accessibility, the application could be deployed on cloud platforms that support Python-based web applications.

Result (Output Image)



Conclusion

The developed voice assistant web application effectively integrates voice interaction, rule-based responses, and AI fallback capabilities within a modern web interface. By leveraging the Web Speech API and the OpenRouter API, the application provides a user-friendly and versatile way to interact with an intelligent assistant for both simple and complex queries. The smooth UI animations further enhance the overall user experience.

FUTURE SCOPE

Discuss potential enhancements and expansions for the system. This could include incorporating additional data sources, optimizing the algorithm for better performance, and expanding the system to cover multiple cities or regions. Consider the integration of emerging technologies such as edge computing or advanced machine learning techniques.

References

- Flask Documentation
- Web Speech API Documentation
- OpenRouter API Documentation
- HTML Standard
- CSS Specifications
- JavaScript Documentation

Thank you