Government Schemes Voice Assistant Documentation

Overview

The **Government Schemes Voice Assistant** is an interactive, multilingual voice-based application designed to assist users in accessing information about government schemes in India. It supports **English**, **Hindi**, and **Hinglish**, allowing users to interact naturally through voice or text input. The system leverages advanced natural language processing (NLP), speech recognition, text-to-speech (TTS), and Retrieval-Augmented Generation (RAG) to provide accurate and context-aware scheme recommendations based on user queries and personal details (e.g., occupation, location).

This documentation provides a comprehensive guide to the system's architecture, functionality, and usage, tailored for sharing with clients to demonstrate the system's capabilities and technical implementation.

Key Features

1. **Multilingual Support**: Handles English, Hindi, and Hinglish for seamless user interaction.
2. **Voice and Text Input**: Supports both voice (via microphone) and text-based input for flexibility.
3. **Speech Recognition**: Uses Google Speech Recognition and Whisper for robust speech-to-text conversion.
4. **Text-to-Speech (TTS)**: Employs gTTS for natural-sounding speech output in multiple languages.
5. **RAG-based Scheme Matching**: Utilizes advanced NLP and embeddings to match user queries with relevant government schemes from a CSV database.
6. **Context-Aware Responses**: Incorporates user details (name, occupation, location) to personalize scheme suggestions.
7. **Language Detection**: Automatically detects the user's preferred language based on input patterns.
8. **Translation**: Supports translation between English, Hindi, and Hinglish for consistent communication.
9. **Error Handling**: Robust fallback mechanisms for speech, TTS, and component failures.
10. **Logging**: Comprehensive logging for debugging and monitoring system performance.

System Architecture

The application is modular, with distinct components handling specific functionalities. Below is an overview of the key modules and their roles:

1. config.py

- **Purpose**: Central configuration file defining system parameters and multilingual phrases.
- **Key Components**:

    - `CONFIG`: Dictionary containing settings like CSV file path, audio timeout, confidence thresholds, TTS model, and more.
    - `PHRASES`: Dictionary of multilingual prompts (English, Hindi, Hinglish) for greetings, queries, and responses.
- **Usage**: Provides consistent configuration across modules and user-friendly prompts for interaction.

2. language_detection.py

- **Purpose**: Detects the language of user input (English, Hindi, or Hinglish).
- **Key Features**:
    - Uses FastText (`lid.176.bin`) for language detection.
    - Includes regex-based Hindi detection and keyword-based Hinglish/English scoring.
    - Fallback mechanisms for robust detection even if FastText fails.
- **Dependencies**: `fasttext`, `re`, `os`.

3. main.py

- **Purpose**: Entry point of the application, initializing and running the voice assistant.
- **Key Features**:
    - Checks for required dependencies (`gtts`, `speech_recognition`, `torch`, `transformers`, `pandas`, `numpy`).
    - Initializes the `VoiceAssistant` class and starts the conversation loop.
    - Handles graceful exits and fatal errors.
- **Dependencies**: `voice_assistant`, `sys`, `os`.

4. scheme_matcher.py

- **Purpose**: Matches user queries to relevant government schemes using a RAG-based approach.
- **Key Features**:
    - Loads and processes scheme data from a CSV file (`Government_schemes_final_english.csv`).
    - Uses SentenceTransformer (`paraphrase-multilingual-MiniLM-L12-v2`) for semantic search.
    - Implements keyword-based search as a fallback.
    - Supports caching of embeddings to improve performance.
    - Incorporates user context (occupation, location) for personalized results.
    - Comprehensive category detection for farmers, fishermen, health, women, education, and business.
- **Dependencies**: `pandas`, `numpy`, `sentence_transformers`, `pickle`, `hashlib`, `logging`.

5. speech_module.py

- **Purpose**: Handles speech recognition for voice input.
- **Key Features**:

    - Supports Google Speech Recognition and Whisper (`openai/whisper-tiny`) for speech-to-text.
    - Multiple microphone initialization strategies for robustness.
    - Configurable settings for energy threshold, pause threshold, and timeout.
    - Fallback to text input if voice recognition fails.
- **Dependencies**: `speech_recognition`, `torch`, `transformers`, `subprocess`, `tempfile`, `numpy`.

6. tts_module.py

- **Purpose**: Converts text responses to speech output.
- **Key Features**:
    - Uses gTTS for text-to-speech conversion.
    - Supports multiple audio players (`mpg123`, `ffplay`, `paplay`) for compatibility.
    - Aggressive text cleaning to ensure smooth TTS output.
    - Prevents duplicate speech calls for better performance.
- **Dependencies**: `gtts`, `subprocess`, `tempfile`, `re`, `logging`.

7. translation.py

- **Purpose**: Translates text between English, Hindi, and Hinglish.
- **Key Features**:
    - Uses `indictrans2` for machine translation.
    - Includes a fallback dictionary for basic translations if the model fails.
    - Handles Hinglish by translating English words to Hindi while preserving Hindi words.
- **Dependencies**: `indictrans2`, `re`.

8. voice_assistant.py

- **Purpose**: Orchestrates the conversation flow and integrates all modules.
- **Key Features**:
    - Manages language selection, user name collection, and context gathering.
    - Parses occupation and location for personalized scheme matching.
    - Formats and delivers scheme responses in the user's preferred language.
    - Handles conversation flow with a limited number of queries (default: 3).
    - Robust error handling and logging.
- **Dependencies**: All other modules (`config`, `tts_module`, `speech_module`, `language_detection`, `translation`, `scheme_matcher`).

Approach to Scheme Data Processing

To enhance the system's ability to provide accurate and context-aware responses based on user voice inputs, we are exploring two approaches for processing data from the CSV file (`Government_schemes_final_english.csv`):

1. **LLM Prompting**: Utilizing large language models (LLMs) with carefully crafted prompts to interpret user queries and match them with scheme data from the CSV. This approach leverages the LLM's natural language understanding to generate relevant responses.
2. **LLM Training on Custom Data**: Training an LLM on the CSV data to fine-tune its ability to understand and respond to queries specific to government schemes. This involves creating a custom dataset from the CSV and fine-tuning the model to improve accuracy.

We are evaluating both methods to determine which provides the best performance in terms of accuracy, response time, and user experience. The most effective approach will be implemented as the primary method for scheme matching and response generation.

Usage Example

1. **Launch**: Run `python main.py`.
2. **Language Selection**:

   ```
   भाषा चुनें / Select Language:
   हिंदी (Hindi)
   English
   हिंग्लिश (Hinglish)
   ```

   User says: "Hinglish" → System responds: "Aapne Hinglish select kiya hai। Chaliye continue karte hain।"
3. **Name Collection**:
   System: "Aapka naam kya hai?"
   User: "Rahul"
   System: "Thank you, Rahul।"
4. **Occupation/Location**:
   System: "Apna occupation ya location batayiye better schemes suggest karne ke liye।"
   User: "Main kisan hoon, Uttar Pradesh se।"
   System parses: Occupation = "farmer", Location = "Uttar Pradesh".
5. **Query**:
   System: "Main government schemes ke baare mein aapki kaise help kar sakta hoon?"
   User: "Kisan ke liye yojana batao।"
   System: "PM Kisan Samman Nidhi। Benefit: Rs 6000 per year।"
6. **Exit**:
   User: "Bas, thank you।"
   System: "Humse contact karne ke liye thank you, Rahul। Aapka din shubh ho!"

Technical Details

- **Language Detection**: Combines FastText with keyword and regex-based detection for accuracy.
- **Scheme Matching**: Uses a hybrid approach (semantic + keyword search) with category-based filtering (e.g., farmer, health, women).
- **TTS Optimization**: Cleans text to avoid encoding issues and ensures smooth playback.

- **Error Handling**: Fallbacks for speech recognition (text input), TTS (print output), and translation (dictionary-based).
- **Performance**: Caches scheme embeddings to reduce computation time on subsequent runs.
- **Logging**: Detailed logs for debugging and monitoring system behavior.

Limitations

1. **Dependency on External Services**: Google Speech Recognition requires internet connectivity.
2. **Microphone Dependency**: Voice input requires a functional microphone and audio setup.
3. **Language Detection**: May struggle with ambiguous Hinglish inputs without clear Hindi/English markers.