Chatbot with Google Gemini AI

The system uses the following **libraries and tools**:

1.LangChain - framework for integrating LLMs with utility tools such as text chunkers, vector stores, and retrievers.

2.Chroma - vector database for efficient storage and retrieval of embeddings (dense representations of text).

3.Google Generative AI:

Embeddings: Converts text into semantic embeddings for similarity search.

Chat Model: Generates conversational responses with safety and respect guidelines.

4.RecursiveCharacterTextSplitter - Splits large documents into smaller, coherent chunks for processing.

5.PyPDFLoader -Specialized loader for extracting text from PDF documents.

Features

1. PDF Processing - Extracts text from PDF files and processes it for AI consumption.

2. Text Chunking -Divides the text into manageable, semantically meaningful units.

3. Embeddings - Converts text chunks into dense vectors to capture semantic meaning.

4. Similarity Search - Retrieves relevant chunks based on user queries using a vector database.

5. Conversational AI - Generates human-like responses, ensuring they are context-aware and safe.

How It Works

1. **PDF Loading** - The PDF is read using PyPDFLoader, Text is extracted and cleaned for further processing.
2. **Text Chunking** - The text is divided into smaller chunks using RecursiveCharacterTextSplitter, ensuring meaningful divisions.
3. **Embedding Creation** - Each chunk is transformed into a dense vector representation using Google Generative AI embeddings.
4. **Vector Store** - The embeddings are stored in Chroma, a vector database, enabling efficient similarity search.

5. **Query Handling** - User queries are compared with the stored embeddings using MultiQueryRetriever to retrieve the most relevant text chunks.
6. **Response Generation** - Retrieved chunks and the query are passed to the Google Generative AI chat model, which generates a detailed response.
7. **Safety Features** - The system includes safety settings to filter harmful or inappropriate content.

Speech-to-Text Application

This application converts audio into text using WhisperX, a fast and efficient extension of OpenAI's Whisper. It supports high-quality transcription with optional GPU acceleration.

Libraries Used

1.**WhisperX** - Transcribes audio into text efficiently, Supports GPU for faster processing.

2.**Google Colab Tools** - Allows access to Google Drive for storing and loading audio files.

How It Works

1.**Install WhisperX** - Ensures the library is ready for transcription tasks.

2.**Mount Google Drive** - Provides access to the audio file stored in Drive.

3.Load the Model - Loads the small model for fast transcription.

4.Transcribe Audio - Converts the audio file into text and timestamps.

5.**Display Results** - Extracts the text from the result object for easy readability.

Frontend

ChatInterface.tsx - A React-based component that provides a chat interface for sending queries to the backend and displaying responses.

Key Features

1. **User Input**: Allows users to type a question or select a predefined sample question.
2. **Chat UI**: Displays messages (from both user and bot) with a scrolling interface.
3. **Backend Integration**: Sends user queries to the backend for processing.

Code Explanation

- **State Management**:
  - messages: Array of chat messages.
  - input: Current user input.
  - isLoading: Indicates whether a backend request is in progress.
- **Main Components**:
  - **Chat Header**: Displays the title and a close button.
  - **Message Area**: Shows the conversation between user and bot.
  - **Input Form**: Allows users to type and send messages.
  - **Sample Questions**: Quick options for user queries.
- **Backend Communication**:
  - Sends user queries to the backend (http://localhost:5000/chat) via a POST request.
  - Handles errors with toast notifications.

Key Functions

- **handleSubmit(e)**:
  - Sends the user's query to the backend.
  - Updates the chat with the bot's response.
- **scrollToBottom()**:
  - Automatically scrolls to the latest message.

Dependencies

- React for building the UI.
- lucide-react for icons.
- Custom UI components like Button, Input, ScrollArea.

How It Works

1. User types a question and clicks "Send."
2. The app sends the query to the backend.
3. The response is displayed in the chat area.
4. Sample questions provide quick options for users.

Backend

app.py - Provides an API endpoint to handle transcription and translation requests.

Key Features

1. **Speech-to-Text Transcription** - Uses Whisper to transcribe audio files into English text.
2. **Translation** - Uses NLLB-200 to translate English text into Telugu.
3. **Chunking** - Handles long text by splitting it into smaller chunks for processing.

Code Explanation

- **Endpoints**:
  - **/chat (POST)** - Accepts a question or transcription request, Returns the transcription or translated response.
- **Components**:
  - **Whisper Model** - Transcribes audio files into English text.
  - **NLLB-200 Model** - Translates English text to Telugu.

Key Functions

- **split_text_into_chunks(text, max_tokens, tokenizer)** - Splits long text into manageable chunks for translation.
- **translate_large_text(text, translation_function, tokenizer)** - Translates text by processing it in chunks.
- **english_to_telugu(text)** - Translates English text into Telugu using NLLB-200.
- **transcribe_and_translate(audio_path)** - Handles the full transcription and translation pipeline:
    1. Transcribes audio into English.
    2. Translates English text into Telugu.

Dependencies

- whisper: For speech-to-text transcription.
- transformers: For handling NLLB-200 translation.
- google.colab: For accessing audio files in Google Drive (specific to Colab).

How It Works

1. An audio file is transcribed into English text using Whisper.
2. The user selects a translation option (currently only English-to-Telugu).
3. The text is translated and returned to the frontend.

Documentation for English-to-Telugu Translation Code
Overview

This code enables the translation of English text into Telugu using the Meta NLLB-200 model, a multilingual translation model designed for low-resource languages. It supports the handling of long text by splitting it into smaller, manageable chunks and translating each chunk sequentially.

Key Features

1. Translation - Converts English text into Telugu using the NLLB-200 model.
2. Chunking - Handles long text by splitting it into smaller pieces for processing.
3. Multilingual Support - Leverages NLLB-200, which is capable of translating between 200+ languages.
4. Pre-trained Model - Utilizes a pre-trained transformer model for high-quality translation.

Workflow

1. Input English Text - Accepts text in English as input.
2. Tokenization and Chunking - Splits long text into smaller chunks to comply with token limits.
3. Translation - Translates each chunk into Telugu.
4. Output Telugu Text - Combines translated chunks into the final Telugu output.

How to Use

1. Install the required libraries, including Hugging Face's transformers.
2. Load the pre-trained NLLB-200 model and tokenizer.
3. Call the main translation function with the English text.
4. Receive the Telugu translation as output.

Dependencies

- Transformers: For loading and running the NLLB-200 translation model.
- PyTorch: For model computations