Documentation for app.py - Dental Clinic Receptionist AI Assistant

Overview

The app.py file implements an AI-powered dental clinic receptionist that interacts with users via voice. It listens to user speech, transcribes it, generates a response, and converts the response to audio. The application uses free, open-source tools to ensure no paid services are required. It is designed to be lightweight and compatible with Docker for deployment.

Features

- **Speech-to-Text**: Captures user speech using a microphone and transcribes it with Google's Speech-to-Text API via the speech_recognition library.
- **Text Generation**: Generates responses using Hugging Face's distilgpt2 model, a lightweight transformer model.
- **Text-to-Speech**: Converts text responses to audio using gTTS (Google Text-to-Speech).
- **Audio Playback**: Plays audio responses using pygame, which is Docker-compatible.
- **Error Handling**: Robustly handles transcription, generation, and playback errors, with graceful shutdown on interrupt.
- **Free Tools**: Uses only free libraries and services, requiring no API keys or paid subscriptions.

Dependencies

- Python 3.10 or 3.11 (Python 3.13.3 may have compatibility issues with PyTorch)
- Libraries:
    - speechrecognition>=3.10.0: For speech-to-text transcription.
    - pyaudio>=0.2.14: For microphone input.
    - transformers>=4.35.0: For text generation with distilgpt2.
    - gTTS>=2.3.2: For text-to-speech conversion.
    - pygame>=2.5.2: For audio playback.
    - python-dotenv>=1.0.0: For loading environment variables (optional, as no API keys are needed).
    - torch>=2.4.1: Required by transformers for model execution.

Setup Instructions

1. Install Python

- **Recommended Version**: Python 3.10 or 3.11 (due to compatibility with PyTorch and transformers).
- **Windows**:
    - Download and install Python 3.11 from python.org.
    - Ensure pip and python are added to your PATH during installation.
- **Verify Installation**:
- python --version
- pip --version

2. Create a Virtual Environment

- Create a virtual environment to isolate dependencies:
- python -m venv venv
- Activate the virtual environment:
- .\venv\Scripts\Activate.ps1

3. Install Dependencies

- Create a requirements.txt file with the following content:
- speechrecognition>=3.10.0
- pyaudio>=0.2.14
- transformers>=4.35.0
- gTTS>=2.3.2
- pygame>=2.5.2
- python-dotenv>=1.0.0
- torch>=2.4.1
- Install dependencies:
- pip install -r requirements.txt
- **Note**: If torch installation fails with Python 3.13.3, switch to Python 3.11 and recreate the virtual environment.

4. Create .env File (Optional)

- No API keys are required, but the code checks for a .env file. Create an empty file or skip this step:

- # .env
- # No API keys required

5. Test Microphone and Audio

- Ensure your microphone and speakers are working (test with a tool like Audacity).
- Verify internet connectivity for Google Speech-to-Text.

Code Explanation

Imports

```
import os
from dotenv import load_dotenv
import speech_recognition as sr
from transformers import pipeline
from gtts import gTTS
import pygame
import tempfile
import time
```

- os: For file operations (e.g., deleting temporary files).
- dotenv: Loads environment variables (optional).
- speech_recognition: Captures and transcribes audio.
- transformers: Generates text responses using distilgpt2.
- gTTS: Converts text to speech.
- pygame: Plays audio files.
- tempfile: Creates temporary files for audio.
- time: Adds delays for error handling.

Environment Setup

```
load_dotenv()
```

- Loads environment variables from .env. No variables are required, so this is a no-op if the file is absent.

AI_Assistant Class

The core class that manages the assistant's functionality.

__init__

```
def __init__(self):
    self.recognizer = sr.Recognizer()
    self.microphone = sr.Microphone()
    self.text_generator = pipeline("text-generation", model="distilgpt2")
    pygame.mixer.init()
    self.full_transcript = [
        {"role": "system", "content": "You are a receptionist at a dental clinic. Be resourceful and efficient."},
    ]
```

- Initializes the speech recognizer and microphone.
- Sets up the distilgpt2 model for text generation.
- Initializes pygame for audio playback.
- Creates a conversation history (full_transcript) with a system prompt defining the assistant's role.

start_transcription

```
def start_transcription(self):
    print("Listening... Speak now.")
    with self.microphone as source:
        self.recognizer.adjust_for_ambient_noise(source)
        while True:
            try:
                audio = self.recognizer.listen(source, timeout=5, phrase_time_limit=10)
                try:
                    transcript = self.recognizer.recognize_google(audio)
                    if transcript.strip():
                        print(f"\nPatient: {transcript}\n")
                        self.generate_ai_response(transcript)
```

```
                else:
                    print("No speech detected, continuing to listen...")
            except sr.UnknownValueError:
                print("Could not understand audio, continuing to listen...")
            except sr.RequestError as e:
                print(f"Google API error: {e}, continuing to listen...")
        except KeyboardInterrupt:
            print("\nStopping transcription...")
            break
        except Exception as e:
            print(f"Error during transcription: {e}")
            time.sleep(1)
```

- Listens for audio input in a loop.
- Adjusts for ambient noise to improve transcription accuracy.
- Captures audio with a 5-second timeout and 10-second phrase limit.
- Transcribes audio using Google's Speech-to-Text API.
- Handles errors (unintelligible audio, API failures, or other exceptions).
- Calls generate_ai_response for valid transcriptions.
- Exits on KeyboardInterrupt (Ctrl+C).

generate_ai_response

```
def generate_ai_response(self, transcript):
    self.full_transcript.append({"role": "user", "content": transcript})
    prompt = f"System: You are a receptionist at a dental clinic. Be resourceful and efficient.\nPatient: {transcript}\nReceptionist:"
    try:
        response = self.text_generator(prompt, max_length=100, num_return_sequences=1, truncation=True)[0]["generated_text"]
        ai_response = response.split("Receptionist:")[-1].strip()
        self.full_transcript.append({"role": "assistant", "content": ai_response})
```

```
        self.generate_audio(ai_response)
    except Exception as e:
        print(f"Error generating AI response: {e}")
```

- Appends the user's transcript to the conversation history.
- Creates a prompt for distilgpt2 with the system role and user input.
- Generates a response (max 100 tokens) and extracts the assistant's part.
- Appends the response to the conversation history.
- Calls generate_audio to convert the response to speech.
- Handles errors during generation.

generate_audio

```
def generate_audio(self, text):
    print(f"\nAI Receptionist: {text}")
    try:
        tts = gTTS(text=text, lang="en")
        with tempfile.NamedTemporaryFile(delete=False, suffix=".mp3") as temp_audio_file:
            tts.save(temp_audio_file.name)
            temp_audio_file_path = temp_audio_file.name
        pygame.mixer.music.load(temp_audio_file_path)
        pygame.mixer.music.play()
        while pygame.mixer.music.get_busy():
            pygame.time.Clock().tick(10)
        pygame.mixer.music.unload()
        os.remove(temp_audio_file_path)
    except Exception as e:
        print(f"Error generating or playing audio: {e}")
```

- Prints the assistant's response.
- Converts text to speech using gTTS and saves it as a temporary MP3 file.
- Loads and plays the audio using pygame.

- Waits for playback to finish.
- Unloads the audio and deletes the temporary file.
- Handles errors during audio generation or playback.

__del__

```
def __del__(self):
    pygame.mixer.quit()
```

- Cleans up pygame resources when the object is destroyed.

Main Execution

```
if __name__ == "__main__":
    greeting = "Thank you for calling Vancouver dental clinic. My name is Sandy, how may I assist you?"
    try:
        ai_assistant = AI_Assistant()
        ai_assistant.generate_audio(greeting)
        ai_assistant.start_transcription()
    except KeyboardInterrupt:
        print("\nShutting down...")
```

- Defines a greeting message.
- Initializes the assistant, plays the greeting, and starts transcription.
- Handles KeyboardInterrupt for clean shutdown.

Running the Application

Locally

1. **Set Up Environment**:
    - Ensure Python 3.10 or 3.11 is installed.
    - Create and activate a virtual environment:
    - python -m venv venv
    - .\venv\Scripts\Activate.ps1
2. **Install Dependencies**:

- Create requirements.txt (see above).
- Install dependencies:
- pip install -r requirements.txt
- If torch installation fails, try:
- pip install torch==2.4.1
- If issues persist, switch to Python 3.11 and recreate the virtual environment.

3. **Run the Application**:
   - Ensure your microphone and speakers are working.
   - Run the script:
   - python app.py
   - The assistant will play a greeting, then listen for your speech. Speak clearly, and it will transcribe, respond, and play the response.
4. **Stop the Application**:
   - Press Ctrl+C to stop the application.

In Docker

1. **Create Dockerfile**:
2. # Use a Python base image
3. FROM python:3.10-slim
4.
5. # Set working directory
6. WORKDIR /app
7.
8. # Install system dependencies for audio (pygame, pyaudio)
9. RUN apt-get update && apt-get install -y \
10. libsndfile1 \
11. libportaudio2 \
12. portaudio19-dev \
13. && rm -rf /var/lib/apt/lists/*

14.
15. # Copy requirements file
16. COPY requirements.txt .
17.
18. # Install Python dependencies
19. RUN pip install --no-cache-dir -r requirements.txt
20.
21. # Copy the application code
22. COPY app.py .
23.
24. # Command to run the application
25. CMD ["python", "app.py"]
26. **Create requirements.txt** (see above).
27. **Build the Docker Image**:
28. docker build -t dental-ai-assistant .
29. **Run the Docker Container**:
    - Ensure audio and microphone access:
    - docker run --rm -it --device=/dev/snd --network=host dental-ai-assistant
    - On Windows, you may need to run in WSL2 or adjust audio device access:
    - # In WSL2
    - wsl docker run --rm -it --device=/dev/snd --network=host dental-ai-assistant
30. **Stop the Container**:
    - Press Ctrl+C to stop the container.

Troubleshooting

Installation Issues

- **PyTorch Installation Fails**:
    - If pip install torch fails with Python 3.13.3, switch to Python 3.11:
    - # Install Python 3.11 via python.org or pyenv

    - python -m venv venv
    - .\venv\Scripts\Activate.ps1
    - pip install -r requirements.txt
    - Try a specific PyTorch version:
    - pip install torch==2.4.1
    - Check PyTorch's installation guide.
- **Dependency Conflicts**:
    - Ensure all dependencies are compatible by using the versions in requirements.txt.
    - If conflicts occur, recreate the virtual environment and reinstall.

Runtime Issues

- **No Transcription**:
    - Verify your microphone is working (test with Audacity).
    - Ensure pyaudio is installed:
    - pip install pyaudio
    - Check internet connectivity for Google Speech-to-Text.
    - If Google API fails repeatedly, it may be due to usage limits. Try again later.
- **No Audio Playback**:
    - Ensure pygame is installed and speakers are working.
    - In Docker, verify --device=/dev/snd is passed:
    - docker run --rm -it --device=/dev/snd --network=host dental-ai-assistant
- **Poor AI Responses**:
    - distilgpt2 is lightweight and may produce less coherent responses. Switch to gpt2-medium for better quality (requires more memory):
    - self.text_generator = pipeline("text-generation", model="gpt2-medium")
    - Fine-tuning the model is possible but requires additional setup.
- **Docker Issues**:
    - If audio or microphone access fails, try --network=host or additional device permissions.

- Ensure libportaudio2 and portaudio19-dev are installed in the container.

Limitations

- **Google Speech-to-Text**: Requires internet access and has usage limits. Responses may fail if limits are exceeded.
- **distilgpt2**: Limited conversational ability compared to advanced models like GPT-3.5. Suitable for simple receptionist tasks.
- **Real-Time Performance**: Slight delays may occur due to Google API latency or model inference.
- **Python Compatibility**: Python 3.13.3 may cause issues with PyTorch or transformers. Use Python 3.10 or 3.11 for stability.

Example Usage

1. Run the application:
2. python app.py
3. Hear the greeting: "Thank you for calling Vancouver dental clinic. My name is Sandy, how may I assist you?"
4. Speak a query (e.g., "I need to book an appointment").
5. The assistant transcribes your speech, generates a response (e.g., "Sure, I can help with that. When would you like to come in?"), and plays it back.
6. Press Ctrl+C to stop.

Future Improvements

- **Better Transcription**: Integrate a local speech-to-text model (e.g., Vosk) for offline support.
- **Improved Responses**: Use a larger model like gpt2-medium or fine-tune distilgpt2 for better coherence.
- **Multi-Turn Conversation**: Enhance full_transcript to maintain context over multiple interactions.
- **Docker Audio**: Streamline audio configuration for Windows and non-WSL2 environments.