End-to-End AI Voice Assistance Pipeline :

Project Overview:

This project aims to create a complete pipeline that converts audio files into text using Whisper, generates a response using a transformer-based language model, converts the response text into speech using tools like Edge TTS, and then plays the generated audio.

This guide explains each component of the project, including installation, usage, and key functions like why i select such libraries , models and code execution .

Installation:

Ensure you have Python installed on your system. The following packages are required:

Libraries and their install command:

!pip install git+https://github.com/openai/whisper.git

!pip install torch

!pip install transformers

!pip install pydub

!pip install edge-tts

!pip install nest_asyncio

!pip install ipython

Why i use these libraries:

1) Whisper : whisper is easy to use for convert voice into text it has simple syntax and it is widely use in industry to convert voice into text with high accuracy.

2) Pytorch : PyTorch is a deep learning library used for building and training machine learning models. In this project PyTorch is used to work with the Whisper model from the `whisper` library and to handle tensors for audio processing and model inference. PyTorch provides the necessary functionality to manipulate data and perform computations required by the models.

3) pydub : Pydub is a library for simple and easy manipulation of audio files.In this project, `pydub` is used for processing audio files, including conversion between different formats (e.g., from MP3 to WAV). It simplifies the task of handling audio file operations and supports various audio formats. For instance, you might use `pydub` to load, convert, and save audio files in different formats for use with text-to-speech (TTS) and speech-to-text (STT) models.

4) nest_asyncio : `nest_asyncio` is a library used to allow nested use of asyncio event loops. In this project, `nest_asyncio` is used to manage asynchronous tasks and run asynchronous functions in environments that already have an event loop running (e.g., Jupyter notebooks). It helps avoid runtime errors when running asynchronous code in these environments by allowing the event loop to be nested.

5) Transformers : transformers is library which is use to import pre trained

   Llm model to give input as a text and generate response from the llm into text format.

6) Edge-tts : edge-tts is use to convert text into voice format which has features to generate voice of male and female both, in this project it convert generate response by llm into voice .

7) ipython : in this project ipython use to play audio file.

Usage :

1. Transcription Using Whisper :

```
import whisper
import torchaudio


# Load the Whisper model
model = whisper.load_model("base")
# Load your .m4a audio file
waveform, sample_rate = torchaudio.load("/content/voice_msg/llm_input_1.m4a")
# Resample the audio to 16000 Hz if needed
resampler = torchaudio.transforms.Resample(orig_freq=sample_rate, new_freq=16000)
waveform = resampler(waveform)


# Convert waveform to mono (Whisper expects mono audio)
if waveform.shape[0] > 1:
    waveform = waveform.mean(dim=0, keepdim=True)
# Convert waveform tensor to numpy array for Whisper
waveform = waveform.squeeze().numpy()
# Transcribe the audio waveform
result = model.transcribe(waveform)
# Print the transcription result
print(result["text"])
```

2) Generating Response Using Transformers :

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM


# Load the pre-trained model and tokenizer

tokenizer = AutoTokenizer.from_pretrained("gokaygokay/Lamini-Prompt-Enchance-Long")

model =
AutoModelForSeq2SeqLM.from_pretrained("gokaygokay/Lamini-Prompt-Enchance-Long")


# Function to generate response using the LLM

def generate_response(input_text, max_length=500):

    # Tokenize the input text

    inputs = tokenizer.encode(input_text, return_tensors="pt")

    # Generate response from the model

    outputs = model.generate(inputs, max_length=max_length, num_return_sequences=1)

    # Decode the generated tokens to text

    response = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # Split the response into sentences

    sentences = response.split('.')

   # Return only the first two sentences

    return '. '.join(sentences[:2]).strip() + '.'


input_text = result["text"]

response_text = generate_response(input_text)

print("LLM Response:", response_text)
```

3) Text-to-Speech Conversion Using Edge TTS:

```
import edge_tts

import asyncio

import nest_asyncio


# Apply nest_asyncio to avoid RuntimeError

nest_asyncio.apply()


async def text_to_speech(text, output_file, voice="en-US-AriaNeural"):

    communicate = edge_tts.Communicate(text, voice)

    await communicate.save(output_file)

    print(f"Audio saved to {output_file}")


# Example usage

text = response_text

output_file = "output_audio.mp3"


# Run the async function in the current event loop

await text_to_speech(text, output_file)
```

4) Playing the Audio Using IPython :

```
from IPython.display import Audio, display


# Play the audio file
def play_audio(file_path):
    display(Audio(file_path, autoplay=True))


# Example usage
output_file = "output_audio.mp3"
play_audio(output_file)
```