Python Code for Jupyter Notebook

Note: This program allows users to paste transcripts from deliberations as well as record deliberations to generate policy recommendations. It also allows you to ask queries regarding the transcript, summary and policy recommendations. You will need to use your own OpenAI API Key and hugging faces key to operate the program

```
import sounddevice as sd
import numpy as np
import scipy.io.wavfile as wav
import threading
import time
import openai
from pyannote.audio import Pipeline
from pydub import AudioSegment
from IPython.display import display, clear_output
import ipywidgets as widgets

# Set your OpenAI API key
openai.api_key = "enter-your-api_key"

# Global variables
recording = False
audio_data = []
start_time = None

# Recording Functions
def callback(indata, frames, time, status):
    if recording:
        audio_data.append(indata.copy())

def record_audio():
    global recording, audio_data, start_time
```

```
    start_time = time.time()
    recording = True
    audio_data = []
    with sd.InputStream(callback=callback, channels=1, samplerate=44100):
        while recording:
            sd.sleep(100)

def start_recording(b):
    global recording
    recording = True
    thread = threading.Thread(target=record_audio)
    thread.start()
    with output_transcription:
        clear_output(wait=True)
        print("Recording started...")

def stop_recording(b):
    global recording
    recording = False
    with output_transcription:
        clear_output(wait=True)
        elapsed_time = time.time() - start_time
        print(f"Recording stopped. Elapsed time: {elapsed_time:.2f} seconds")

        # Save the recording
        filename = "conversation.wav"
        audio_np = np.concatenate(audio_data, axis=0)
        wav.write(filename, 44100, (audio_np * 32767).astype(np.int16))
        print(f"Recording saved as {filename}")
        return filename
```

```
# Process and Transcribe
def process_audio(file_path):
    pipeline = Pipeline.from_pretrained("pyannote/speaker-diarization-3.0",
use_auth_token="enter-your-hugging -faces-token")

    audio = AudioSegment.from_wav(file_path)
    audio.export(file_path, format="wav")

    diarization = pipeline(file_path)

    segments = []
    with open(file_path, "rb") as audio_file:
        for turn, _, speaker in diarization.itertracks(yield_label=True):
            segments.append((turn.start, turn.end, speaker))

    transcript = []
    for start, end, speaker in segments:
        segment = audio[start * 1000:end * 1000]
        segment.export("temp_segment.wav", format="wav")

        with open("temp_segment.wav", "rb") as segment_file:
            response = openai.Audio.transcribe("whisper-1", segment_file)
            transcript.append(f"{speaker}: {response['text']}")

    return "\n".join(transcript)
# Global variables to store the summary, policies, and transcribed conversation
summary_and_policies = ""
transcribed_conversation = ""

# Summarize Text
def summarize_text(text):
```

```
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are the smartest policy analyst in the world and a helpful assistant."},
            {"role": "user", "content": f"Please provide a summary for the following text and tell me the key theme of the text and suggest 3 conservative policies and 3 liberal policies:\n\n{text}"}
        ],
        max_tokens=800
    )
    # Store the generated summary and policies
    global summary_and_policies
    summary_and_policies = response['choices'][0]['message']['content'].strip()
    return summary_and_policies

# Handle Queries
def handle_query(b):
    query = query_box.value
    if query.strip() == "":
        with output_query:
            clear_output(wait=True)
            print("Please enter a valid query.")
        return

    # Generate response for the user's query using the summary, policy recommendations, and query
    with output_query:
        clear_output(wait=True)
        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=[
```

```
            {"role": "system", "content": "You are the smartest policy analyst in the world and a helpful assistant."},
            {"role": "user", "content": f"Here is the transcribed conversation:\n{transcribed_conversation}\n\nSummary and policy recommendations:\n{summary_and_policies}\n\nNow, please answer the following query: {query}"}
        ],
        max_tokens=300
    )
    reply = response['choices'][0]['message']['content'].strip()
    print(f"Answer to your query:\n{reply}")

# Analyze Recording and Summarize
def analyze_recording(b):
  global transcribed_conversation
  with output_transcription:
    clear_output(wait=True)
    filename = stop_recording(None)
    transcribed_conversation = process_audio(filename)
    print("Transcription:\n", transcribed_conversation)
    summary = summarize_text(transcribed_conversation)
    print("Summary:\n", summary)

    # Enable the query box after displaying the summary
    query_box.disabled = False
    query_button.disabled = False

# New Function to Paste Transcribed Conversation
def analyze_pasted_transcription(b):
  global transcribed_conversation
  with output_transcription:
    clear_output(wait=True)
```

```
    transcribed_text = pasted_text.value
    if transcribed_text:
        transcribed_conversation = transcribed_text  # Store the pasted conversation
        print("Processing pasted transcription...")
        summary = summarize_text(transcribed_conversation)
        print("Summary:\n", summary)

        # Enable the query box after displaying the summary
        query_box.disabled = False
        query_button.disabled = False
    else:
        print("No text was pasted.")

# Setup Buttons and Textbox for Pasting Transcription
start_button = widgets.Button(description="Start Recording")
start_button.on_click(start_recording)

stop_button = widgets.Button(description="Stop Recording")
stop_button.on_click(analyze_recording)

pasted_text = widgets.Textarea(placeholder="Paste transcribed conversation here...",
layout=widgets.Layout(width='100%', height='200px'))
analyze_pasted_button = widgets.Button(description="Analyze Pasted Text")
analyze_pasted_button.on_click(analyze_pasted_transcription)

# Query Input and Button
query_box = widgets.Textarea(placeholder="Enter your query here...",
layout=widgets.Layout(width='100%', height='100px'), disabled=True)
query_button = widgets.Button(description="Ask Query", disabled=True)
query_button.on_click(handle_query)

# Separate output boxes
```

output_transcription = widgets.Output() # For transcription and summary

output_query = widgets.Output() # For query results

Display Buttons, Textbox, and Outputs

display(start_button, stop_button, pasted_text, analyze_pasted_button, output_transcription, query_box, query_button, output_query)

Screenshots

1]:

```python
import sounddevice as sd
import numpy as np
import scipy.io.wavfile as wav
import threading
import time
import openai
from pyannote.audio import Pipeline
from pydub import AudioSegment
from IPython.display import display, clear_output
import ipywidgets as widgets

# Set your OpenAI API key
openai.api_key = [redacted]

# Global variables
recording = False
audio_data = []
start_time = None

# Recording Functions
def callback(indata, frames, time, status):
    if recording:
        audio_data.append(indata.copy())

def record_audio():
    global recording, audio_data, start_time
    start_time = time.time()
    recording = True
    audio_data = []
    with sd.InputStream(callback=callback, channels=1, samplerate=44100):
        while recording:
            sd.sleep(100)
```

```python
def start_recording(b):
    global recording
    recording = True
    thread = threading.Thread(target=record_audio)
    thread.start()
    with output_transcription:
        clear_output(wait=True)
        print("Recording started...")

def stop_recording(b):
    global recording
    recording = False
    with output_transcription:
        clear_output(wait=True)
        elapsed_time = time.time() - start_time
        print(f"Recording stopped. Elapsed time: {elapsed_time:.2f} seconds")

        # Save the recording
        filename = "conversation.wav"
        audio_np = np.concatenate(audio_data, axis=0)
        wav.write(filename, 44100, (audio_np * 32767).astype(np.int16))
        print(f"Recording saved as {filename}")
        return filename

# Process and Transcribe
def process_audio(file_path):
    pipeline = Pipeline.from_pretrained("pyannote/speaker-diarization-3.0", use_auth_token=[redacted])

    audio = AudioSegment.from_wav(file_path)
    audio.export(file_path, format="wav")

    diarization = pipeline(file_path)

    segments = []
    with open(file_path, "rb") as audio_file:
        for turn, _, speaker in diarization.itertracks(yield_label=True):
            segments.append((turn.start, turn.end, speaker))
```

```python
    transcript = []
    for start, end, speaker in segments:
        segment = audio[start * 1000:end * 1000]
        segment.export("temp_segment.wav", format="wav")

        with open("temp_segment.wav", "rb") as segment_file:
            response = openai.Audio.transcribe("whisper-1", segment_file)
            transcript.append(f"{speaker}: {response['text']}")

    return "\n".join(transcript)
# Global variables to store the summary, policies, and transcribed conversation
summary_and_policies = ""
transcribed_conversation = ""

# Summarize Text
def summarize_text(text):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are the smartest policy analyst in the world and a helpful assistant."},
            {"role": "user", "content": f"Please provide a summary for the following text and tell me the key theme of the text and suggest 3 conservati
        ],
        max_tokens=800
    )
    # Store the generated summary and policies
    global summary_and_policies
    summary_and_policies = response['choices'][0]['message']['content'].strip()
    return summary_and_policies

# Handle Queries
def handle_query(b):
    query = query_box.value
    if query.strip() == "":
        with output_query:
            clear_output(wait=True)
            print("Please enter a valid query.")
        return
```

```
    # Generate response for the user's query using the summary, policy recommendations, and query
    with output_query:
        clear_output(wait=True)
        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "system", "content": "You are the smartest policy analyst in the world and a helpful assistant."},
                {"role": "user", "content": f"Here is the transcribed conversation:\n{transcribed_conversation}\n\nSummary and policy recommendations:\n
            ],
            max_tokens=300
        )
        reply = response['choices'][0]['message']['content'].strip()
        print(f"Answer to your query:\n{reply}")

# Analyze Recording and Summarize
def analyze_recording(b):
    global transcribed_conversation
    with output_transcription:
        clear_output(wait=True)
        filename = stop_recording(None)
        transcribed_conversation = process_audio(filename)
        print("Transcription:\n", transcribed_conversation)
        summary = summarize_text(transcribed_conversation)
        print("Summary:\n", summary)

        # Enable the query box after displaying the summary
        query_box.disabled = False
        query_button.disabled = False
```

```
# New Function to Paste Transcribed Conversation
def analyze_pasted_transcription(b):
    global transcribed_conversation
    with output_transcription:
        clear_output(wait=True)
        transcribed_text = pasted_text.value
        if transcribed_text:
            transcribed_conversation = transcribed_text  # Store the pasted conversation
            print("Processing pasted transcription...")
            summary = summarize_text(transcribed_conversation)
            print("Summary:\n", summary)

            # Enable the query box after displaying the summary
            query_box.disabled = False
            query_button.disabled = False
        else:
            print("No text was pasted.")

# Setup Buttons and Textbox for Pasting Transcription
start_button = widgets.Button(description="Start Recording")
start_button.on_click(start_recording)

stop_button = widgets.Button(description="Stop Recording")
stop_button.on_click(analyze_recording)

pasted_text = widgets.Textarea(placeholder="Paste transcribed conversation here...", layout=widgets.Layout(width='100%', height='200px'))
analyze_pasted_button = widgets.Button(description="Analyze Pasted Text")
analyze_pasted_button.on_click(analyze_pasted_transcription)

# Query Input and Button
query_box = widgets.Textarea(placeholder="Enter your query here...", layout=widgets.Layout(width='100%', height='100px'), disabled=True)
query_button = widgets.Button(description="Ask Query", disabled=True)
query_button.on_click(handle_query)

# Separate output boxes
output_transcription = widgets.Output()  # For transcription and summary
output_query = widgets.Output()  # For query results
```

```
# Display Buttons, Textbox, and Outputs
display(start_button, stop_button, pasted_text, analyze_pasted_button, output_transcription, query_box, query_button, output_query)
```

Start Recording

Stop Recording

Mr. Stéphane Bergeron:

By way of introduction, I want to make three comments, which I hope will be rather brief, before I get into the substance of the matter and explain why we will be voting against this bill at report stage. Here is my first comment.

When he asked his question, my colleague from Berthier—Maskinongé did a great job explaining why we are voting against this bill at report stage. We voted in favour of this bill in principle because we support the idea of having tighter controls on imports coming in from forced labour, slavery and child labour. However, as my Conservative colleague noted, as we listened to some of the witnesses we realized that this bill has major flaws. As the member who introduced it admitted, this is a bill that simply encourages transparency, essentially relies on corporate goodwill, and does not provide for the necessary checks or for what we call due diligence. As my colleague from Berthier—Maskinongé noted, the government will not necessarily follow up to ensure that goods produced from forced labour or child labour are indeed not imported into Canada. I think that is a major flaw of this bill.

My second introductory comment is simple: I believe that the sponsors of this bill, Senator Miville-Dechêne and the member for Scarborough—Guildwood have very good intentions. I believe that their reasons for introducing this bill are honourable. They put their heart and soul into the bill and worked very hard on it. I believe they deserve our utmost respect for the work that has been done to date, but it is unfortunately not

Analyze Pasted Text

Processing pasted transcription...
Summary:
Summary:
The debate in the House of Commons focused on Bill S-211, which aims to address forced labour and child labour in global supply chains. Various members of different parties expressed their opinions on the bill, highlighting the importance of tighter controls on imports coming from forced labour, slavery, and child labour. While some members expressed support for the bill and its intention of promoting transparency in supply chains, others raised concerns about the bill's effectiveness and called for more rigorous measures to hold companies accountable for human rights violations. The key theme of the text is the need to address forced labour and child labour in global supply chains through legislative action.

Key Theme:
The key theme of the text is the need for legislation to address forced labour and child labour in global supply chains, emphasizing the importance of transparency, accountability, and enforcement mechanisms to combat human rights violations in the production of goods.

Conservative Policies:
1. Stricter enforcement of laws to combat forced labour and child labour in supply chains.
2. Implementing financial penalties for companies found to be using forced labour or child labour.
3. Advocating for criminal liability for non-compliance with regulations related to forced labour and child labour.

Liberal Policies:
1. Introducing legislation to eradicate forced labour from Canadian supply chains.
2. Strengthening regulations to ban goods produced using forced labour.
3. Increasing transparency and due diligence requirements for companies to prevent human rights abuses in their supply chains.

Enter your query here...

Ask Query