06-17-2025: YouTube Speaker Diarizer and Summarizer

YouTube Speaker Diarizer: A Full Pipeline to Extract & Summarize Speaker Insights from Any Video

In this article, I'll walk through the complete design, implementation, and usage of my latest project: a YouTube Speaker Diarizer and Summarizer. This pipeline takes any YouTube video link, separates the speakers, transcribes their speech, and summarizes what each person said individually. It was built using Hugging Face, OpenAI, and Python, and runs entirely from the terminal.

This tool is perfect for:

- Breaking down interviews, debates, and podcast episodes
- Getting a speaker-by-speaker summary of long content
- Quickly extracting viewpoints from talk shows, panels, or multi-host content

---

🛠️Project Objectives

- Accept a YouTube URL input
- Download and convert the video to clean WAV audio
- Identify individual speakers using a diarization model
- Transcribe each speaker's audio segment using Whisper
- Generate summaries of each speaker's thoughts using GPT-4
- Print clean outputs of who said what

💡 Tech Stack Overview

Component	Tool/Library	Purpose
Video Download	`yt-dlp`	Downloads and converts YouTube audio
Audio Transcription	`Whisper (openai/whisper-large-v2)`	Transcribes audio to text
Speaker Diarization	`pyannote.audio`	Distinguishes between different speakers

Component	Tool/Library	Purpose
Summarization	`openai.ChatCompletion (GPT-4)`	Summarizes transcripts speaker-wise
Audio Manipulation	`pydub`	Splits audio by speaker timestamp
Env Management	`python-dotenv`, `.env`	Securely loads API tokens
Runtime	`torch`	Handles CUDA/CPU inference switching

📂Folder Structure

```
youtube_diarizer_06_17_2025/
├── summarize_youtube.py       # Main script file
├── audio.wav                  # Output audio (auto-generated)
├── audio/                     # Temporary folder (cleaned up)
├── .env                       # Contains HuggingFace & OpenAI tokens
└── /venv (yt_diarizer)        # Conda environment folder
```

🔐.env Format

Purpose: Securely load your secret API keys for HuggingFace and OpenAI.

```
HF_TOKEN=your_huggingface_token_here
OPENAI_API_KEY=your_openai_key_here
```

Make sure this is created in your project root. Tokens are retrieved from:

- Hugging Face: https://huggingface.co/settings/tokens
- OpenAI: https://platform.openai.com/account/api-keys

⚖️Setup Instructions

Step-by-step to install everything you need:

```
# Create a new Conda environment
conda create -n yt_diarizer python=3.10 -y
conda activate yt_diarizer
```

```
# Install all necessary packages
pip install yt-dlp ffmpeg-python python-dotenv torch torchaudio pyannote.audio
scipy openai transformers pydub
```

```
# Make sure ffmpeg works
# Either add to system PATH or provide full path to yt-dlp if needed
```

```
# Run the summarizer on any YouTube video link
python summarize_youtube.py "https://www.youtube.com/watch?v=your_video_id"
```

🧰Full Pipeline Breakdown

1. Download YouTube Audio

Purpose: Pulls highest-quality audio and converts to `.wav` for processing.

```
from yt_dlp import YoutubeDL

def download_audio(youtube_url, output_path="audio"):
    ydl_opts = {
        "format": "bestaudio/best",
        "outtmpl": output_path,
        "postprocessors": [
            {"key": "FFmpegExtractAudio", "preferredcodec": "wav"},
        ],
    }
    with YoutubeDL(ydl_opts) as ydl:
        ydl.download([youtube_url])
    return output_path
```

2. Diarize Audio with Pyannote

Purpose: Breaks audio into speaker-specific segments.

```
from pyannote.audio import Pipeline
import torch

def load_diarizer():
    pipe = Pipeline.from_pretrained("pyannote/speaker-diarization-3.1",
use_auth_token=HF_TOKEN)
    return pipe.to(torch.device("cuda" if torch.cuda.is_available() else "cpu"))
```

3. Slice Audio by Segment

Purpose: Extracts only the portion of audio spoken by a specific speaker.

```
from pydub import AudioSegment

audio = AudioSegment.from_wav("audio.wav")
segment = audio[start_ms:end_ms]
segment.export("tmp_chunk.wav", format="wav")
```

4. Transcribe with Whisper

Purpose: Converts sliced audio into text using Whisper via Hugging Face.

```
from transformers import pipeline

asr = pipeline("automatic-speech-recognition", model="openai/whisper-large-v2")
result = asr("tmp_chunk.wav", return_timestamps=True)
text = result["text"]
```

5. Summarize Transcription with GPT-4

Purpose: Condenses full speech into a digestible paragraph per speaker.

```
import openai
openai.api_key = OPENAI_API_KEY

response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[{"role": "user", "content": f"Summarize this person:\n{text}"}],
    temperature=0.4,
```

```
)
summary = response.choices[0].message.content.strip()
```

🔧Sample Output

```
==== SPEAKER_01 ====
The person does not provide any clear arguments or opinions in this text. They
are asking questions about someone else's expensive purchases...

==== SPEAKER_00 ====
This person is discussing their various properties and investments around the
world... Brewery House... restoring UBHL...
```

🕑Challenges Faced

Issue	Fix
Whisper timing bug	Used `return_timestamps=True` and short segments
pyannote model slow on CPU	Added torch.device auto-detection
ChatCompletion call broke	Downgraded to `openai==0.28` to fix v1.0 compatibility issues
Segments transcribing full audio	Fixed by slicing audio using `pydub` before transcription

---

🧪Potential Next Steps

- Save output as Markdown/Text instead of terminal-only
- Add UI (Streamlit or Flask + React)
- Run Whisper & Summarizer async for speed boost
- Auto-process podcast feeds or YouTube playlists
- Add translation or emotion detection per speaker

🕑Final Thoughts

This project was one of the most complete and real-world applicable things I've built in pure Python. The idea that I can paste in a video link, run one command, and get speaker-level summaries of *who said what* is powerful for anyone working in:

- Journalism

- Podcast editing
- Debate analysis
- Academic research
- Content repurposing

This isn’t just a summarizer. It’s a full pipeline that understands speakers, context, and intent.

Drop in a link. Get insights.

📰Why This Project Matters

Most tools out there either summarize entire videos as a blob of text or give rough transcripts without distinguishing who’s actually speaking. This project stands out because it focuses on speaker-by-speaker clarity — helping you understand not just *what* was said, but *who* said it. Whether it's for digesting interviews or breaking down panel discussions, this pipeline bridges a gap that basic transcription and summarization tools still overlook.

Let me know if you want the full code, the zipped project, or to integrate it with your own workflows!