**A Step-By-Step Guide To Summarizing Audio Files in Python**

Speech summarization made easy

As the name suggests, summarization is the process of generating a concise summary of a given piece of information. This information can appear as text, audio, video, pictures, etc. In other words, summarization is the process of selecting/generating relevant pieces of information that are representative of the entire input.

Building a data-driven summarization system is a common task in natural language processing due to its broad downstream applicability in areas like summarizing web pages, long documents, product reviews, etc.

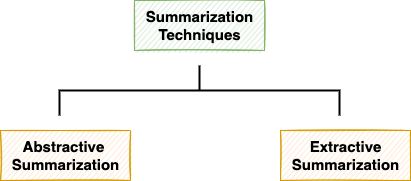
While most of the prior methods in this domain have focused on applying summarization techniques to text, summarizing audio/video files is still a relatively unexplored area in natural language processing.

Therefore, in this post, I will demonstrate how you can build an Audio Summarization system using the **AssemblyAI API** and **Python**.

The highlight of the article is as follows:

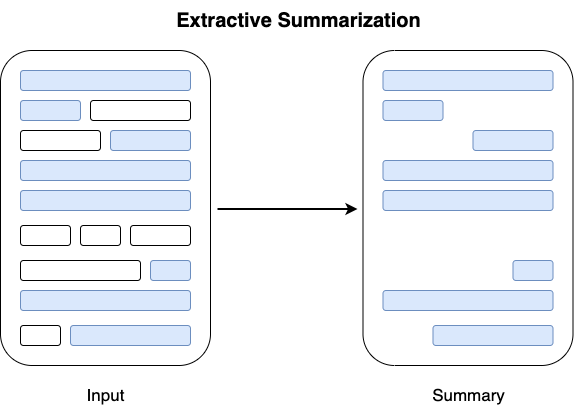
**Introduction to Summarization**

From the technical perspective of building an intelligence-embedded summarization system, the techniques leveraged to solve this problem can be broadly categorized into two categories:



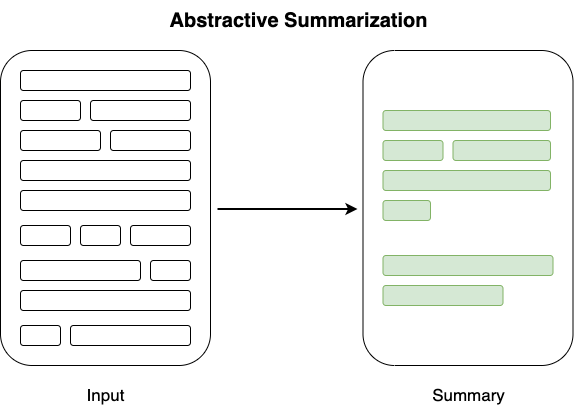
The two categories of summarization techniques

In simple terms, **Extractive Summarization** extracts the top-n sentences that are representative of the given information (text, audio, etc.).



Extractive Summarization (Image by author)

On the other hand, the **Abstractive Summarization** technique generates a summary from scratch using natural language generation methods.

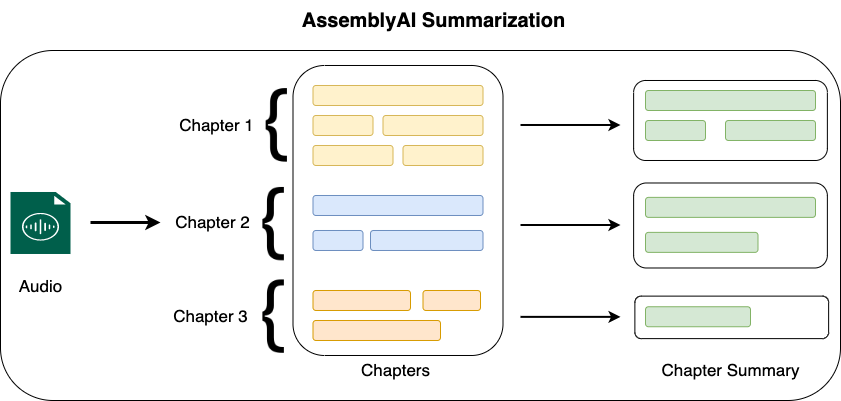


Abstractive Summarization (Image by author)

**Summarizing Audio File**

WE will use of AssemblyAI API to leverage their deep learning models and summarize the input audio file.

The AssemblyAI API internally follows a two-step process for summarizing audio content, as shown below:



Summarization workflow of AssemblyAI API

* First, the audio file is transcribed and broken into individual chapters (or paragraphs).
* Then, the summarization module summarizes individual chapters identified in the input audio/video and returns the chapter-wise audio summary.

The steps to summarize the audio file are demonstrated below:

**Step 1: Get the API Token**

To access the AssemblyAI services, you should first create an account on the [AssemblyAI](https://app.assemblyai.com/signup" \t "_blank) website and get your access token.

**Step 2: Install libraries**

AssemblyAI provides transcription services with API requests. Therefore, to summarize the audio file, you should send the requests from your local computer using the [requests](https://pypi.org/project/requests/) package in python as follows:

**Step 3: Send Transcription Request**

Next, we will start implementing the summarization pipeline using AssemblyAI API to transcribe the audio file.

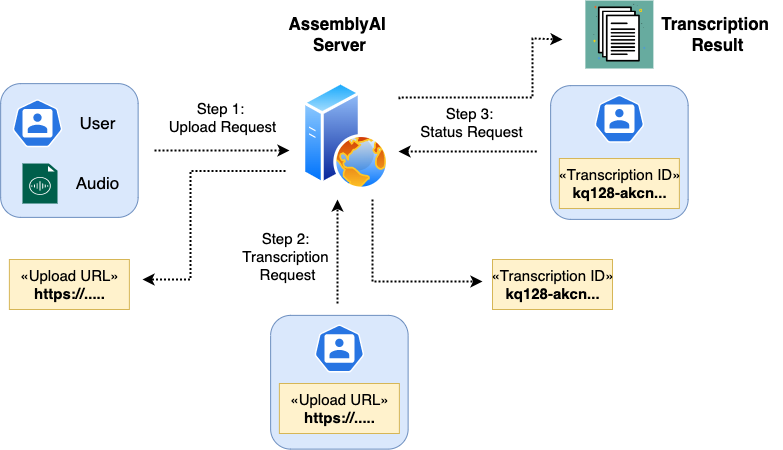
* **Import libraries**

First, we import the following libraries into the python environment:

* **Define the Authorization Key, Headers, and the Transcription Endpoints**

1. You should have obtained the API key from Step 1 above.
2. The upload\_endpoint specifies the service to be invoked to upload the audio file to AssemblyAI’s servers.
3. The transcription service is invoked separately from the upload service, and for this, we will use the transcription\_endpoint to summarize the audio file.

The workflow after this step is depicted in the image below:



Transcription workflow to use the AssemblyAI API

* **Upload the Audio file**

Next, we shall define a method to upload the input audio file to AssemblyAI’s servers. The upload() method is implemented below:

The method opens the audio file located at file\_path and reads it in chunks of 5MBs. Moreover, the function returns the upload\_url obtained from the JSON response.

*Note: It is not necessary to upload the audio file to AssemblyAI as long as it is accessible with a URL. You can upload the audio to the AWS S3 bucket or to audio hosting services like SoundCloud etc.*

* **Transcribe the Audio File**

Now, we will define the transcribe() method to send a POST request to the AssemblyAI API to start the transcription process.

The transcribe()method accepts the upload\_url as an argument and creates a POST request to obtain the unique transcription\_id for our request.

* **Fetch the Result**

To view the transcription result, we should now create a GET request for the AssemblyAI servers. To know the status of our specific request, we shall provide the unique identifier (transcription\_id) in our new GET request. The get\_result() method is implemented below:

As the transcription process may take time, we should make repeated GET requests to know the status. Once the status changes to completed or indicates an error, we will return the JSON response.

* **Execute the pipeline**

Lastly, we execute the entire pipeline by invoking the individual functions created above as follows:

The JSON response can be interpreted as follows:

* We see the **status** of transcription as **completed**, which indicates that the audio file was successfully transcribed.
* The entire transcription can be accessed using the text key of the JSON response as follows:
* We can access the summary of the audio file with the chapters key of the JSON response as follows:

The value corresponding to the chapters key is a list of dictionaries, where each individual dictionary represents the summary of one chapter.

In this case, the number of chapters is 1. Hence, we have obtained the summary for the entire audio file in a single dictionary.

Further, each individual dictionary holds five different keys, which are:

* summary: This represents the summary of the chapter.
* headline: As the name suggests, this represents the essence of the entire chapter in a single line.
* gist: This is an ultra-short summary of the content in the chapter.
* start: This is the starting timestamp (in milliseconds) of the chapter in the input audio.
* end: Finally, this denotes the ending timestamp (in milliseconds) of the chapter in the input audio.

**Analysis**

Once the results are ready, we can perform the following analysis to understand the summary quality quantitatively.

* First, let’s find whether the summary generated by the model is through an extractive or abstractive technique.

As none of the three summary sentences appear in the transcription text, we can conclude that the model performed an **abstractive summarization**.

We can do a similar analysis for the headline and the gist as follows:

* Next, we will compute the word ratio between the summary text and the transcription text. Ideally, the summary should be in the range of 5–10% of the text.

**Conclusion**

To conclude, in this post, we discussed a particular NLP use case of the AssemblyAI API. Specifically, we saw how to build a summarization system on a pre-recorded audio file and interpret the results obtained.