**SPEECH TO TEXT**

This project provides an implementation of a Speech-to-text model using Python and Facebook Wav2Vec2 model with the help of transformers and tokenizers. The system transcribes speech input to text. In our project we take live recording and convert it into the text.



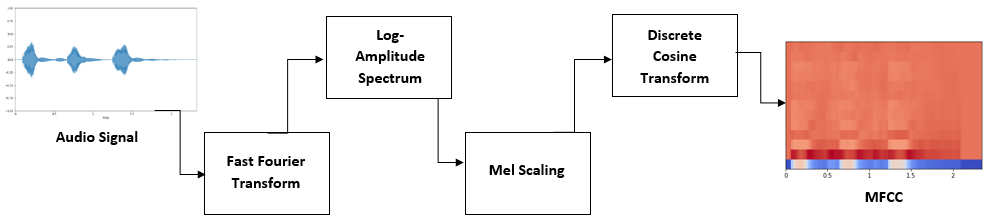
**GETTING STARTED**

Before getting involved in writing the code one should have good knowledge of waves, speech waveforms, and spectrograms. The main component involves Audio Feature extraction, in brief, audio feature extraction can be explained as

Audio feature extraction is the process of extracting relevant information or features from an audio signal that can be used for various applications such as speech recognition, music analysis, and speaker identification. Audio signals are complex waveforms, and their analysis requires techniques that capture the characteristics relevant to the task at hand.

Commonly used audio feature extraction techniques include the Mel-Frequency Cepstral Coefficients (MFCC), which are based on the human auditory system's response to sound frequencies, and the spectrogram, which visualizes the frequency and time components of an audio signal. Other techniques include pitch detection, energy, and zero-crossing rate analysis.

These extracted features are then used as input to machine learning models such as neural networks, which can learn to recognize patterns in the data and perform tasks such as speech recognition or music genre classification. Overall, audio feature extraction plays a crucial role in many audio-related applications and is essential in processing audio signals for machine learning.



(<https://youtube.com/playlist?list=PL-wATfeyAMNqIee7cH3q1bh4QJFAaeNv0>)

A famous playlist one can watch for learning about Audio and its feature extraction.

**RESOLVING MODEL ISSUE**Top of Form

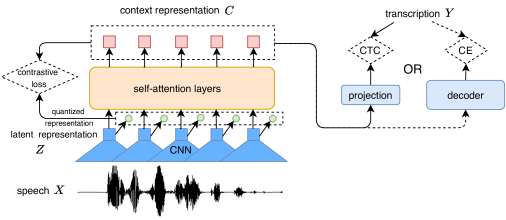
Initially tried to build an ML model like DeepSpeech2. DeepSpeech2 is a speech recognition system that uses a deep neural network and a language model to transcribe speech directly to the text. It employs the Long Short-Term Memory (LSTM) architecture and the Connectionist Temporal Classification (CTC) loss function to improve accuracy. But we faced certain errors while compiling the code that was not able to be resolved.

(NEED SS FROM CODE TO PUT HERE)

Later the solution came up to use a pre-trained model, a famous model named “FACEBOOK WAV2VEC2.0” from Hugging Face has been used (reference link: <https://huggingface.co/facebook/wav2vec2-base> ).

What is Facebook’s Wav2Vec2 model?

Facebook's wav2vec2 is a speech recognition system that uses self-supervised learning to improve accuracy. It pre-trains a neural network on unlabelled speech data and fine-tunes it on a labelled dataset. wav2vec2 uses a contrastive learning framework to learn robust and generalizable representations of speech, which has led to state-of-the-art results on benchmarks. It has also been used in speaker recognition and audio classification.



**Why we chose Facebook Wav2Vec2?**

Here are some factors that might make wav2vec2 a good choice:

1. Self-supervised learning: wav2vec2 uses self-supervised learning, which means it can be trained on large amounts of unlabelled data, making it more efficient and cost-effective than supervised learning-based models.
2. Contrastive learning framework: wav2vec2 uses a contrastive learning framework, which has been shown to be effective in learning robust and generalizable speech representations, leading to improved accuracy.
3. State-of-the-art results: wav2vec2 has achieved state-of-the-art results on several speech recognition benchmarks, indicating its high performance and potential for real-world applications.
4. Pre-trained models available: Facebook has released pre-trained wav2vec2 models that can be fine-tuned on specific tasks, which can save time and resources compared to training a model from scratch.

This model is Transformer based and uses one of its essential thing tokenizers to train the model.

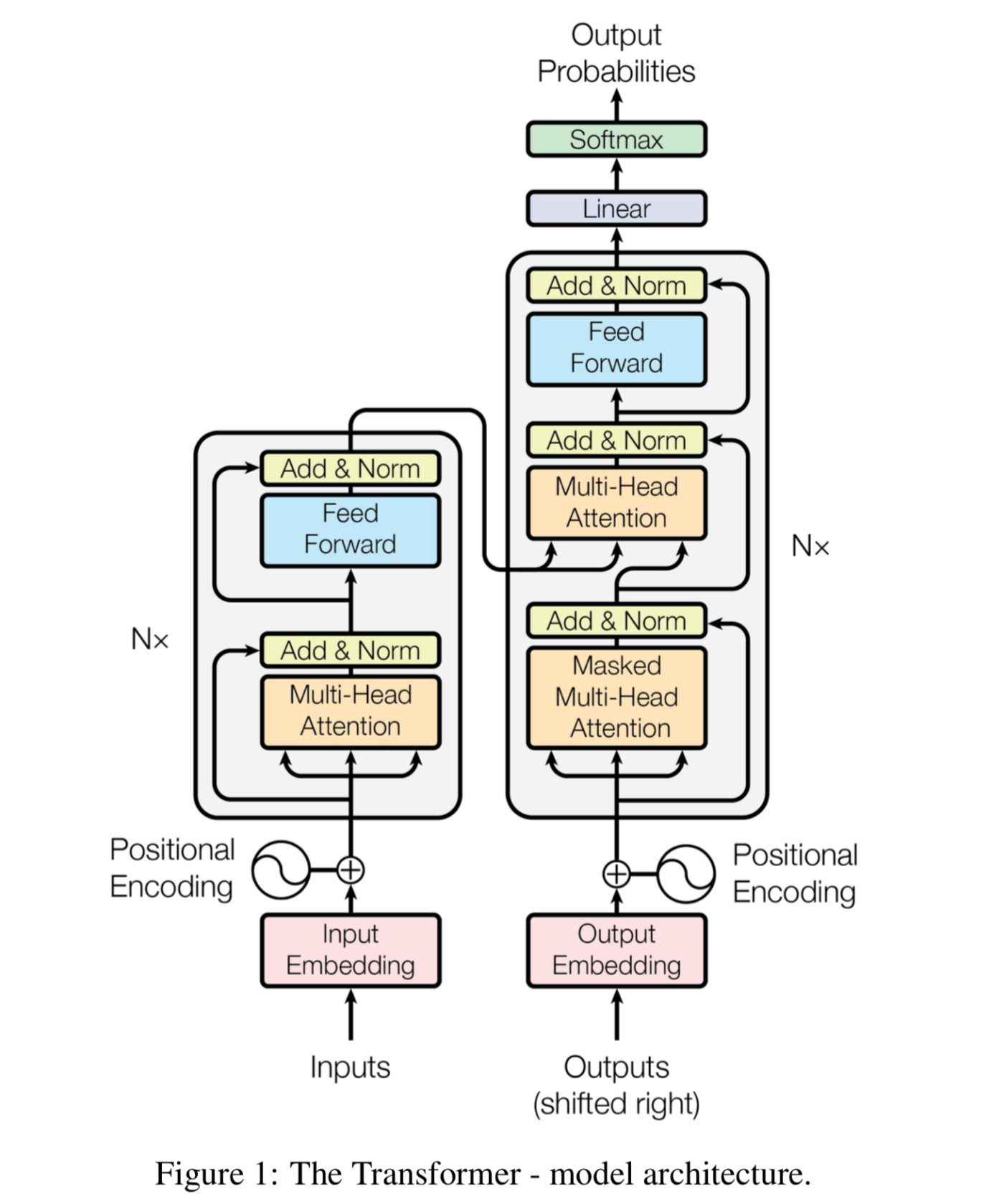
**TRANSFORMERS**

Transformer is deep learning model that adopts the mechanism of self-attention, differently weighing the significance of each part of the input data. It is primarily used in fields of NLP and CV.

Transformer has Encoder-decoder architecture.

Encoder- It transforms the data from one representation to other. It is compressed representation of data which have salient features of data which is more useful for model.

Decoder- It performs reverse operation of Encoder. It decompresses the compressed data into original shape with desired output.



Transformer-based models are increasingly popular in speech recognition tasks due to their effectiveness in capturing long-range dependencies in sequential data.

Here are some reasons why transformer-based models may be a good choice:

1. Attention mechanism: Transformers use an attention mechanism that allows the model to selectively focus on relevant parts of the input sequence, improving its ability to capture long-range dependencies.
2. Parallel processing: Transformers can process sequences in parallel, making them more efficient and faster than traditional recurrent neural network (RNN) models.
3. Transfer learning: Pre-trained transformer-based models such as BERT, GPT, and T5 can be fine-tuned on a variety of downstream NLP tasks, such as text classification, sentiment analysis, and question answering, achieving state-of-the-art results and saving time and resources compared to training a model from scratch.
4. Multimodal processing: Transformers can be used for multimodal processing, combining information from multiple modalities such as text, audio, and visual data, enabling the development of more sophisticated models for tasks such as speech recognition, image captioning, and video analysis.

**TOKENIZERS**

One of the essential components of the pre-trained model used is Tokenizers, these are essential parts for training transformer-based models. Tokenization is the process of breaking down the raw text into smaller units or tokens, which can then be used as input to the transformer model.

Tokenizers perform several important functions in the training of transformer-based models:

1. Breaking down text into tokens: Tokenizers split the text into smaller units such as words, sub-words, or characters, depending on the specific tokenizer used.
2. Encoding tokens: Tokenizers assign a unique numerical representation to each token, which can be used as input to the transformer model.
3. Handling out-of-vocabulary (OOV) words: Tokenizers can handle words that are not in the model's vocabulary by splitting them into sub-words or characters, allowing the model to still process them.
4. Padding and truncating sequences: Tokenizers can add padding tokens to ensure that all input sequences are of the same length, or truncate sequences that are too long.

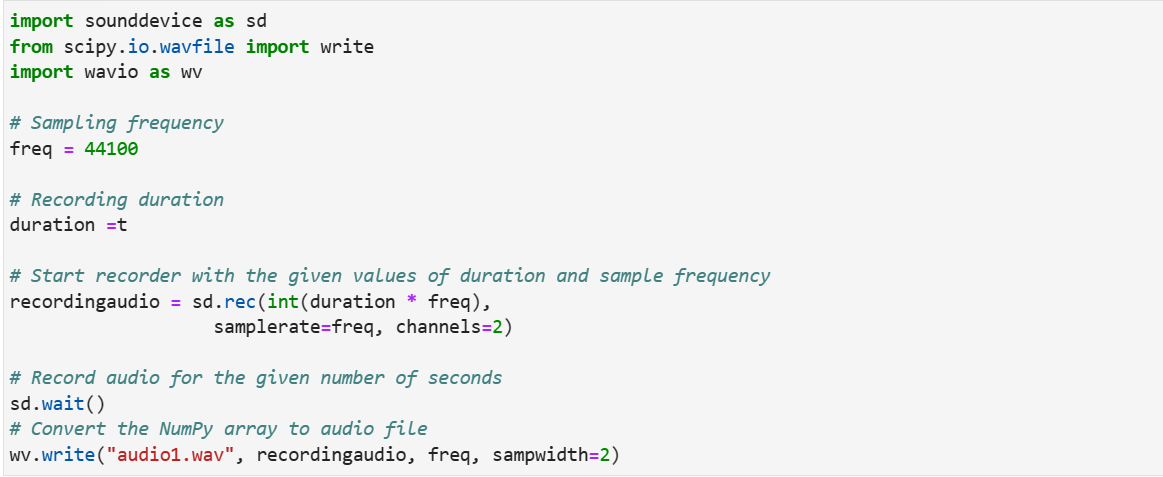
<https://huggingface.co/docs/tokenizers/index>

Reference the link to learn more about tokenizers.

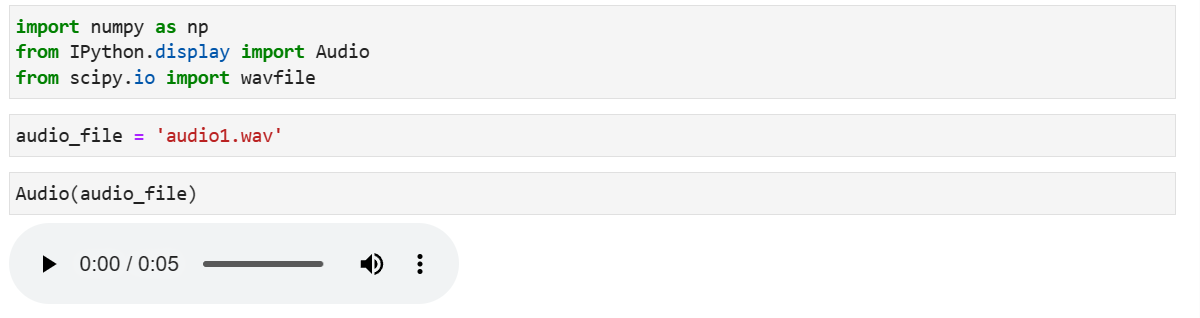
**DATA PRE-PROCESSING**

We chose the pre-trained model which works effectively, being a pre-trained model, we will use live recording clips to convert it into text. We will add synthetic noise to the clips and then use it to get text, we will do denoising too.

AUDIO RECORDING FEAUTRE

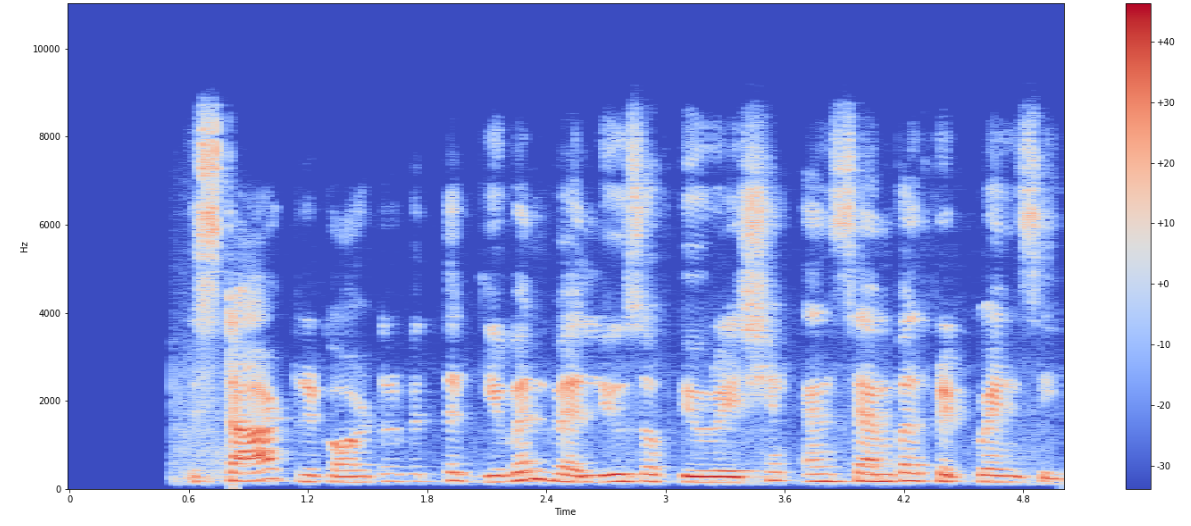


We can record our voice for required duration with the help of microphone available in our devices.



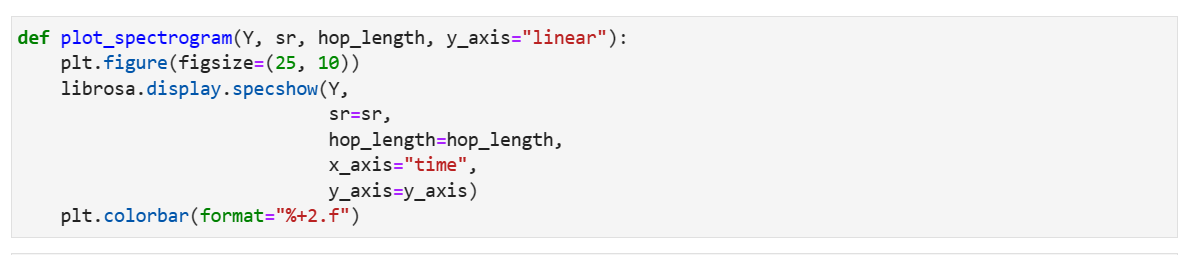
Later we converted recorded file into wav file for further processing.

**PLOTTING SPECTROGRAMS**



A spectrogram is a visual representation of the frequencies that make up a signal or sound over time. It is a way to analyse the frequency content of a signal in a time-varying manner. The x-axis of the spectrogram represents time, and the y-axis represents frequency.

To create a spectrogram, the signal is first broken down into short segments and then a mathematical function called the Fourier transform is applied to each segment. The result of this process is a set of values representing the frequency content of the signal at that point in time.



**NOISE AND DENOISING**

In audio processing, noise generation refers to the process of adding a noise signal to an existing audio file to simulate the effect of background noise or interference.

Noise in audio can come from various sources such as electrical interference, background noise, or even the inherent noise in electronic components used in the recording or playback equipment. This noise can have a detrimental effect on the quality of the audio signal, especially when the signal is of low amplitude.

To simulate the effect of noise in an audio signal, a noise signal is generated and added to the original signal. The noise signal can be generated in different ways depending on the type of noise to be simulated. For example, Gaussian noise can be generated by drawing random values from a Gaussian distribution, while pink noise can be generated using a filter that creates a signal with a spectrum that falls off at a rate of 3 dB per octave.



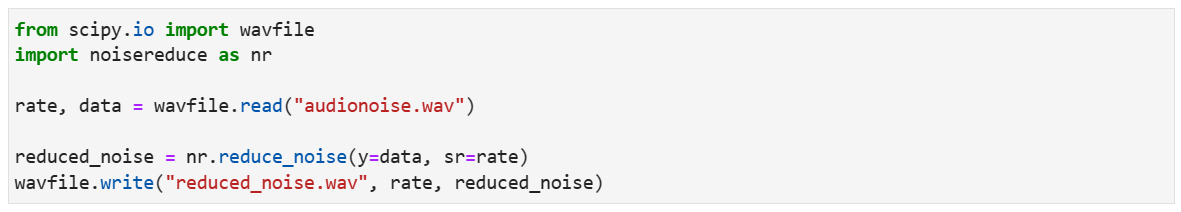
The **np. random. normal** function is used in the context of adding noise to the signal based on a Gaussian distribution.

In the code **noise=np. random. normal (0, RMS, signal. shape[0])**, the **np. random. normal** function is used to generate a random noise signal with a Gaussian distribution. The noise signal is generated with a mean of 0 and a standard deviation of **RMS**, which means that the standard deviation of the noise signal is proportional to the RMS value of the original signal. This is done to ensure that the noise signal has a similar energy level to the original signal.

WHY NORMAL ADDITION?

A normal distribution is a bell-shaped probability distribution that is symmetric around its mean. It is a commonly used distribution in statistics and probability theory due to its mathematical properties and its frequent occurrence in real-world phenomena.

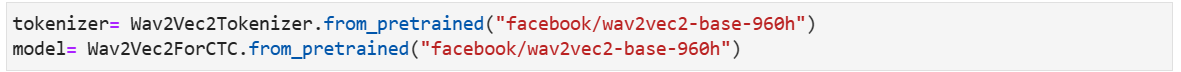
In the context of generating noise for a signal, a normal distribution can be a useful choice because it has several desirable properties. First, it is a continuous distribution, meaning that it can take on any value within a range, which can be important when generating a wide range of noise values for a signal. Second, it has a well-defined mean and variance, which can be used to control the properties of the generated noise signal. Finally, it is a common distribution for modeling real-world noise sources, such as electrical interference or thermal noise.



Later, denoising data using noise reduce inbuilt library. Here, noise reduction is done with one of the algorithms named Spectral Subtraction.

Spectral subtraction is a widely used method for noise reduction in audio signals. It works by estimating the noise power spectral density (PSD) from a noisy signal and subtracting it from the noisy signal's PSD to obtain an estimate of the clean signal's PSD. Spectral subtraction has several advantages over other noise reduction techniques, including its simplicity and computational efficiency. However, it has some limitations, such as the need for accurate noise estimation and the potential for introducing artifacts or distortions in the reconstructed signal. Therefore, spectral subtraction is often used in combination with other noise reduction techniques to achieve better performance.

**USING PRE-TRAINED MODELS**



A screenshot of a computer code

Description automatically generated with low confidence

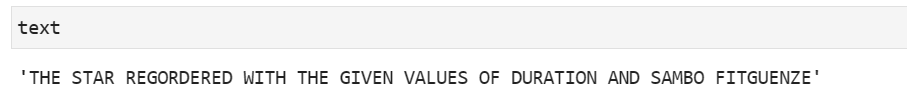
Loading audio using librosa and using pre-defined tokenizers to get desired output.

Here is an overview of what each line of code is doing:

1. **input\_values = tokenizer(input\_audio,return\_tensors="pt").input\_values**: This line uses a tokenizer to convert the input audio file into a format that can be processed by the language model. The tokenizer converts the audio file into a sequence of numerical tokens, which are then wrapped in a PyTorch tensor object. The **return\_tensors="pt"** argument specifies that the output should be a PyTorch tensor object.
2. **logits = model(input\_values).logits**: This line passes the input tensor through a pre-trained language model, which generates a set of logits (unnormalized probability scores) for each token in the input sequence. The **logits** object is a PyTorch tensor that contains the logits for each token in the input sequence.
3. **predicted\_ids = torch.argmax(logits,dim=-1)**: This line applies the **argmax** function to the logits tensor along the last dimension (i.e., the dimension that corresponds to the different possible tokens in the vocabulary), to obtain the predicted token IDs for each position in the input sequence. The **predicted\_ids** object is a PyTorch tensor that contains the predicted token IDs.
4. text = tokenizer.batch\_decode(predicted\_ids)[0]: This line uses the tokenizer to convert the predicted token IDs into text. The batch\_decode function converts a batch of token IDs into a batch of text strings. In this case, since we only have a single input sequence, we access the first element of the batched output (which is a single string). The text object is a Python string that contains the transcribed text from the input audio file.

Overall, this code performs automatic speech recognition (ASR) by transcribing an input audio file into text using a pre-trained language model.

**GETTING OUTPUT**



We will get output in the format of text after we let the wav file to run through pre-trained model.

**TEXT TO SPEECH**

**What is Transformer TTS?**

A transformer TTS (text-to-speech) is a type of speech synthesis system that uses transformer-based neural networks to convert written text into natural-sounding speech.

During the training process, the model learns to generate speech by predicting the acoustic features that correspond to a given sequence of phonemes. This is achieved through a multi-layered neural network that uses self-attention to capture the dependencies between the different elements of the input sequence.

In contrast to traditional TTS systems, which rely on concatenative or statistical parametric methods, transformer-based TTS models generate speech in a more holistic and natural way. They are capable of producing more expressive and nuanced speech by modeling the prosodic features of human speech, such as intonation, rhythm, and stress.

Overall, transformer-based TTS has emerged as a state-of-the-art approach to speech synthesis and is widely used in commercial applications, such as virtual assistants and audio-book narration.

**Why are we using Transformer TTS?**

Transformers are used in text-to-speech (TTS) conversion because they are highly effective at modeling the relationship between input text and corresponding audio output.

The primary goal of TTS is to generate speech that sounds natural and human-like. This requires the conversion of written text into a sequence of phonemes, or the smallest units of sound in a language, which are then combined to create spoken words. Transformers are well-suited for this task because they excel at capturing long-term dependencies between the different elements of the input sequence.

In particular, transformer models have been shown to be highly effective at generating speech from text by employing an attention mechanism that allows the model to focus on relevant parts of the input text during the generation process. This allows the model to capture the subtle nuances of human speech and produce more natural-sounding output.

Overall, the ability of transformers to model complex relationships between input text and corresponding audio output makes them a popular choice for text-to-speech conversion.

**GETTING STARTED**

Installing all the necessary libraries and dependencies.

First, you need to install the necessary libraries and dependencies. You can use pip, the Python package manager, to install these dependencies. Here are the libraries and dependencies you need to install:

•PyTorch

•NumPy

•librosa

•TTS (Transformer TTS library)

1. We need to load the Transformer TTS model. The Transformer TTS model consists of an encoder and a decoder. The encoder takes in the input sequence and produces a sequence of hidden representations that capture the semantic meaning of the input text. The decoder then takes in the hidden representations and generates a sequence of speech spectrograms.
2. Next, we need to load the MelGAN vocoder which will convert the speech spectrograms to audio. The main objective of MelGAN is to generate high-quality audio waveforms from speech spectrograms, which are two-dimensional representations of speech signals. The MelGAN architecture consists of two main components: a generator network and a discriminator network. The generator network takes in a sequence of speech spectrograms and generates a corresponding sequence of audio waveforms. The discriminator network is trained to distinguish between real and synthetic audio waveforms.

**FEATURES**

\*We have also provided options for generating male, female and child voices by varying the sample rate of the audio generated.

\*We have also provided a system where we can listen to the audio at a faster or slower rate.