

Multilingual Speech Recognition Model

TensorGo



March 13, 2024

yedla Karthikeya

**Building Multilingual Speech Recognition Model for RAG without Training**

**Objective**

To build a multilingual speech recognition model without training, using a pre-trained multilingual speech recognition model, such as Multilingual Whisper, to enable RAG to perform tasks in multiple languages.

**Introduction**

Multilingual automatic speech recognition (ASR) models have gained significant interest because of their ability to transcribe speech in more than one language. This is fueled by the growing multilingual communities as well as by the need to reduce complexity. You only need one model to handle multiple languages.

ASR models convert speech to text at a high level.  At inference time, they consume audio files as input and generate text tokens or characters as output. More precisely, at each audio sample timestep, the model outputs the log probability for each one of a total of num\_classes tokens.

At training time, you supply text transcripts as well as audio files as inputs. As the model trains, it uses the transcripts to compute a training loss. It gradually minimizes this loss and improves its weights and gets its output transcripts as close to the originals as possible.

The multilingual context adds a few aspects to the picture. During inference, you typically don’t know the language or languages that the audio contains.  However, if the model knows the language ID (LID) that it encounters in the audio, it may be useful for it to output it.

This could be used to put together language-specific processing pipelines downstream from an ASR model. Similarly, you may need to supply the LID values present in each sample during training.

Code-switching refers to changing between different languages during a conversation.  Such models must anticipate that each sample may contain more than one LID value and need to be trained accordingly.

**There are two basic approaches to creating multilingual models.**

In the first approach, you can largely ignore the fact that there are multiple languages in the dataset and shuffle the transcripts as-is, letting the model figure everything out. If the model uses text tokenization, you only have to ensure that the tokenizer vocabulary size is sufficient to cover all the languages. You combine the transcripts across the different languages and train the tokenizer.

In the second approach, you tag each text sample in the transcript with the appropriate LID.  If the model uses tokenizers, you train them separately on each language and then use the NeMo aggregate tokenizer functionality to combine them.

Each language gets an assigned range of token IDs, and the model learns to generate them.  During decoding, to determine the LID of a particular token, you look at the range of its ID.

**Model – Whisper API**

Whisper is a general-purpose automatic speech recognition model that was trained on a large audio dataset. The model can perform multilingual transcription, speech translation, and language detection.  Whisper can be used as a voice assistant, chatbot, speech translation to English, automation taking notes during meetings, and transcription.

**Transcription** is a process of converting spoken language into text. In the past, it was done manually, and now we have AI-powered tools like Whisper that can accurately understand spoken language.

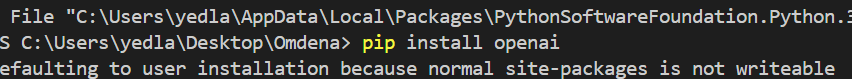
**Languages** supported for transcriptions and translations by OpenAI Whisper API are:

Afrikaans, Arabic, Armenian, Azerbaijani, Belarusian, Bosnian, Bulgarian, Catalan, Chinese, Croatian, Czech, Danish, Dutch, English, Estonian, Finnish, French, Galician, German, Greek, Hebrew, Hindi, Hungarian, Icelandic, Indonesian, Italian, Japanese, Kannada, Kazakh, Korean, Latvian, Lithuanian, Macedonian, Malay, Marathi, Maori, Nepali, Norwegian, Persian, Polish, Portuguese, Romanian, Russian, Serbian, Slovak, Slovenian, Spanish, Swahili, Swedish, Tagalog, Tamil, Thai, Turkish, Ukrainian, Urdu, Vietnamese, and Welsh.

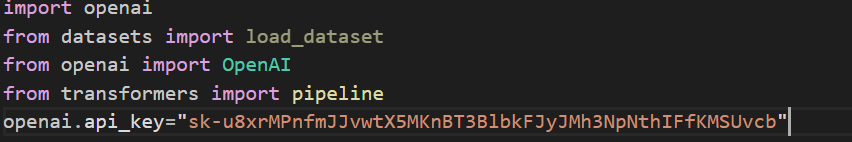
**The file formats** supported by Whisper API are mp3, mp4, mpeg, mpga, m4a, wav, and webm.

**Speech to text with OpenAI Whisper API:**

1. Install OpenAI Python API by using pip

****

1. After that, we have to generate API keys by accessing [OpenAI API](https://platform.openai.com/) webpage
2. You can set up a key within your Python program using openai.api\_key.

****

**Dataset**

English:

We will use few audio clips taken from the NLTM speech data taken from internet.

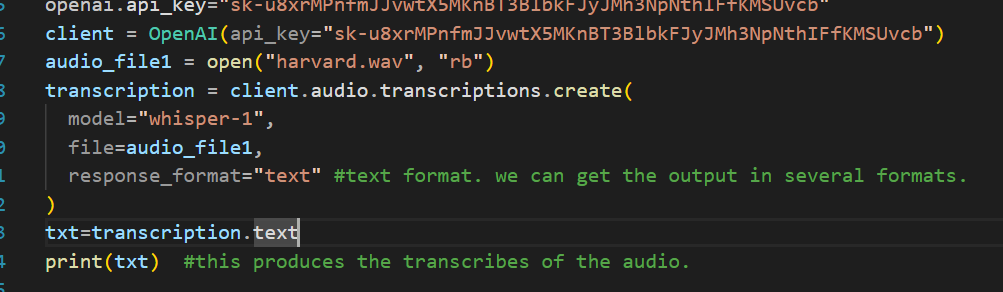
Hindi:

We will use few audio clips taken from the NLTM speech data taken from internet.

**English Transcriptions**

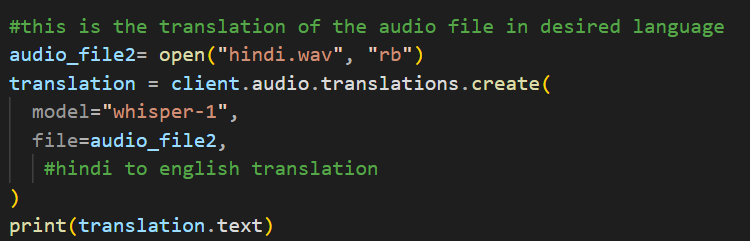
The transcriptions API takes as input the audio file you want to transcribe and the desired output file format for the transcription of the audio. It support multiple input and output file formats. By default, the response type will be json with the raw text included.

The Audio API also allows you to set additional parameters in a request. For example, if you want to set the response\_format as text, our request would look like the following:



**Hindi to English Translations**

The translations API takes as input the audio file in any of the supported languages and transcribes, if necessary, the audio into English. This differs from our /Transcriptions endpoint since the output is not in the original input language and is instead translated to English text.

****

**Summarization:**

For summarization facebook bart model was used.

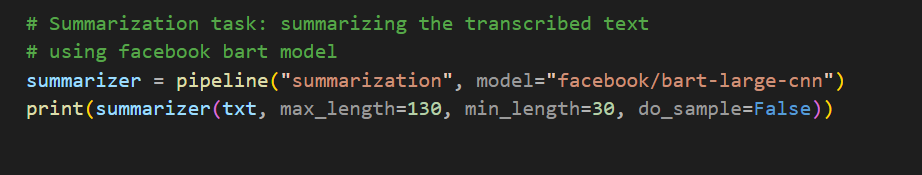
**Model description:**

BART is a transformer encoder-encoder (seq2seq) model with a bidirectional (BERT-like) encoder and an autoregressive (GPT-like) decoder. BART is pre-trained by (1) corrupting text with an arbitrary noising function, and (2) learning a model to reconstruct the original text.

BART is particularly effective when fine-tuned for text generation (e.g. summarization, translation) but also works well for comprehension tasks (e.g. text classification, question answering).

1. Import transformers pipeline





**Model Evaluation Metrics:**

The BLEU score for Translation

Bilingual Evaluation understudy is a metric for automatically evaluating machine-translated text. The BLEU score is a number between zero and one that measures the similarity of the machine-translated text to a set of high quality reference translations. A value of 0 means that the machine-translated output has no overlap with the reference translation (low quality) while a value of 1 means there is perfect overlap with the reference translations (high quality).A value of 0.3 or higher is usually considered good. Python Natural Language Toolkit (NLTK) provides an implementation of the BLEU score.

The formula for BLEU score is as follows:

BLEU = BP \* exp(∑ pn)

BP: Brevity Penalty

pn; pn is the precision of n-grams

ROGUE for Summarization

ROUGE score is a set of metrics commonly used for text summarization tasks, where the goal is to automatically generate a concise summary of a longer text. ROUGE was designed to evaluate the quality of machine-generated summaries by comparing them to reference summaries provided by humans. ROUGE score measures the similarity between the machine-generated summary and the reference summaries using overlapping n-grams, word sequences that appear in both the machine-generated summary and the reference summaries. The most common n-grams used are unigrams, bigrams, and trigrams. ROUGE score calculates the recall of n-grams in the machine-generated summary by comparing them to the reference summaries.

The formula for ROUGE score is as follows:

ROUGE = ∑ (Recall of n-grams)