

OBJECTIVE:

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

BACKGROUND:

RAG is a generative model that can be used for a variety of tasks, including speech recognition, translation, and summarization. However, RAG is currently only trained to perform these tasks in a single language. By building a multilingual speech recognition model without training, we can enable RAG to perform these tasks in multiple languages without the need for additional training.

Solution Approach:

Using a pre-trained model (like Multilingual Whisper), the objective is to build a multilingual speech recognition model and combine it with a RAG (Retrieval-Augmented Generation) model. Without further training, the integrated system should be able to carry out activities like transcription, translation, and multilingual querying of a fictitious document.

Tools and Technologies:

- Multilingual Whisper: A pre-trained multilingual speech recognition model.
- RAG Model: For performing generative tasks based on the transcriptions.
- Python: Main programming language for implementing the project.
- Hugging Face Transformers: Library for accessing pre-trained models.

Multilingual Speech Recognition Model:

An artificial intelligence system that can comprehend and translate spoken words into text in several languages is called a multilingual speech recognition model. These models use cutting-edge methods from the fields of natural language processing and machine learning to identify different dialects, accents, and linguistic subtleties. They are useful for applications like real-time translation tools, multilingual virtual assistants, and automated transcription services since they are trained on a variety of datasets to guarantee high accuracy and versatility.

RAG Model:

A Retrieval-Augmented Generation (RAG) model in multilingual speech recognition combines retrieval-based methods with generative models to improve performance. For the purpose of inference, it retrieves pertinent instances from a database of previously collected multilingual speech data. These illustrations help a generative model to better accurately translate or transcribe incoming speech inputs. RAG models can handle a wide range of languages and accents more skillfully by utilizing both retrieval and generation, which improves the overall accuracy and resilience of multilingual speech recognition systems.

Multilingual Whisper:

Introduction: Whisper is a versatile Transformer-based sequence-to-sequence model designed for various speech processing tasks introduced by Open-ai. Unlike traditional speech processing pipelines, Whisper integrates multiple functionalities such as multilingual speech recognition, speech translation, language identification, and voice activity detection into a single model.

Key features include:

- **Multitask Training:** Tasks like speech recognition, translation, and identification are jointly represented as a sequence of tokens, allowing for streamlined processing.
- **Tokenization and Language Support:** Supports a wide range of languages, managed through task specifiers and classification targets within the sequence.

Technical Setup and Requirements:

To deploy Whisper, ensure the following setup:

- Python 3.8-3.11 and PyTorch 1.10.1 (compatible with recent versions).
- Installation via `pip install -U openai-whisper`, with dependencies including OpenAI's tiktoken for efficient tokenization.

Key Features and Capabilities:

- **Language Understanding:** Multilingual models excel at understanding input text in different languages. They can detect the language being used and generate contextually appropriate responses.
- **Language Generation:** They can generate text in multiple languages based on the input and context provided. This is particularly useful for tasks like translation, where the model can interpret input text in one language and produce corresponding text in another.
- **Cross-Lingual Transfer:** These models can transfer knowledge learned from one language to another. For instance, if the model has been trained extensively on English data, it can still generate coherent responses in languages where data is less abundant, albeit with varying degrees of accuracy depending on the language.

Implementation, Performance, and Practical Usage :

1. Model Sizes and Performance Metrics

Whisper offers various model sizes with differing speed and accuracy trade-offs:

Tiny, Base, Small, Medium, and Large models, each optimized for specific tasks and resource constraints.

2. Command-Line Interface (CLI) Usage

Whisper provides robust CLI commands for seamless integration:

- ✓ **Transcription:** Command examples for transcribing audio files (`whisper audio.wav --model medium`).
- ✓ **Language Specific Options:** Support for specifying languages (`--language Japanese`) and tasks (`--task translate`) to enhance versatility.

3. Python Integration

Integration within Python environments is straightforward:

- ✓ **Model Loading:** `whisper.load_model("base")` to load specific model sizes.
- ✓ **Transcription:** `model.transcribe("audio.mp3")` for transcription tasks within Python scripts.
- ✓ **Advanced Usage:** Utilizing `whisper.detect_language()` and `whisper.decode()` for lower-level model access and manipulation.

Other Key Components:

Transformers

- ❖ **Definition:** Hugging Face created the transformers open-source library, which offers cutting-edge models for natural language processing (NLP) that are built on transformer topologies.

- ❖ **Features:** It comes with pre-trained models for a range of natural language processing (NLP) applications, including language translation, named entity recognition (NER), text categorization, and question answering.
- ❖ **Usage:** Transformers are used by researchers and developers to experiment with novel NLP algorithms, integrate NLP capabilities into applications, and fine-tune pre-trained models on certain tasks.

Librosa:

- ❖ **Definition:** The Python library librosa is used to analyze music and audio. It has functionality for loading audio files, extracting features like Mel spectrograms, pitch estimating, and beat tracking, among other audio signal processing activities.
- ❖ **Applications:** Often utilized in domains such as sound analysis, music information retrieval (MIR), and speech processing. For applications like sound classification, speech recognition, and music recommendation systems, it is indispensable.
- ❖ **Features:** Provides tools for time-frequency analysis, audio feature extraction, and audio data visualization.

Torch:

- ❖ **Definition:** Torch is an open-source machine learning library, a scientific computing framework, and a script language based on the Lua programming language. It provides a wide range of algorithms for deep learning and machine learning and is known for its flexibility and speed.
- ❖ **Applications:** PyTorch is widely used to build neural networks for image recognition, natural language processing, and speech recognition. Leveraged for tasks like object detection, segmentation, and image classification. Utilized for language modeling, translation, sentiment analysis, and text generation.
- ❖ **Features:** Seamless integration with Python, making it easier to use with other Python libraries and tools.

MarianMTModel and MarianTokenizer:

- ❖ **Definition:** **MarianMTModel:** A machine translation model based on the Marian NMT framework. It is part of the Hugging Face Transformers library and provides pre-trained models for various language pairs. **MarianTokenizer:** The tokenizer associated with the MarianMTModel, responsible for preprocessing text into a format suitable for the model, including tokenization and encoding.
- ❖ **Applications:** Generating multilingual data for training other models in natural language processing (NLP). Translating text from one language to another in applications like multilingual chatbots, content localization, and real-time communication tools.
- ❖ **Features:** Access to a wide range of pre-trained models for different language pairs, reducing the need for training from scratch. Extensive support for many language pairs, facilitating global communication and applications.

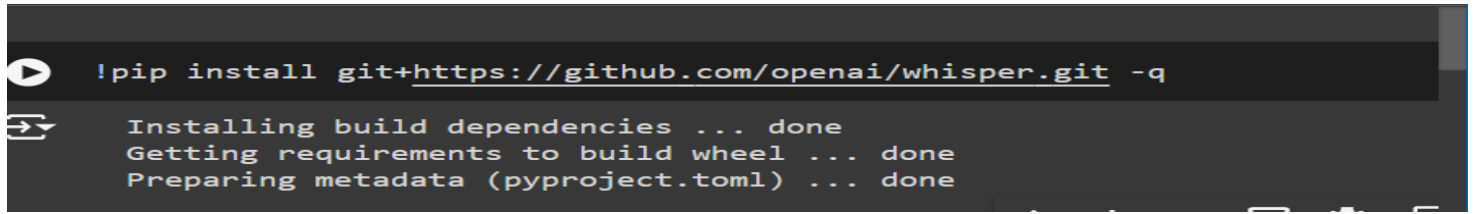
FAISS (Facebook AI Similarity Search):

- ❖ **Definition:** FAISS (Facebook AI Similarity Search) is a library developed by Facebook AI Research designed to efficiently search for similar vectors (high-dimensional data points). It is particularly useful for tasks that involve large-scale similarity search, such as nearest neighbor search, clustering, and recommendation systems.
- ❖ **Applications:**
 - **Nearest Neighbor Search:** Quickly finding the closest points in a high-dimensional space, which is critical for recommendation systems, image retrieval, and more.
 - **Clustering:** Grouping similar data points together, useful in data analysis and machine learning.

❖ Features:

- High Performance: Optimized for speed and scalability, capable of handling large datasets efficiently.
- Versatility: Supports various types of vector data and multiple search algorithms, including exact and approximate nearest neighbor search.

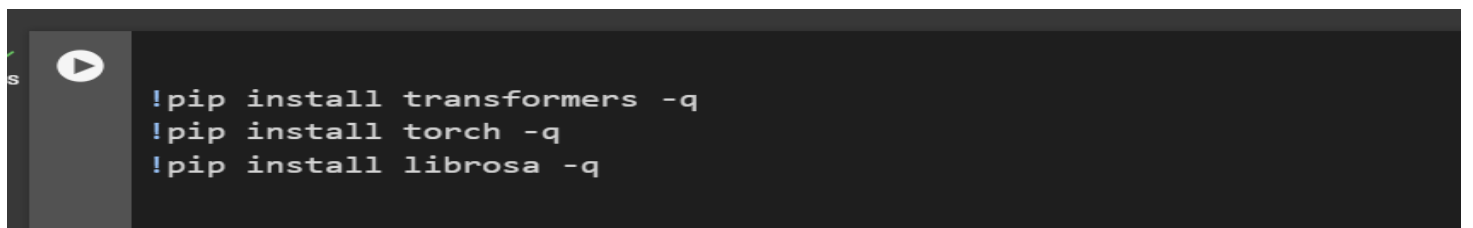
CODE:



```
!pip install git+https://github.com/openai/whisper.git -q
```

Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done

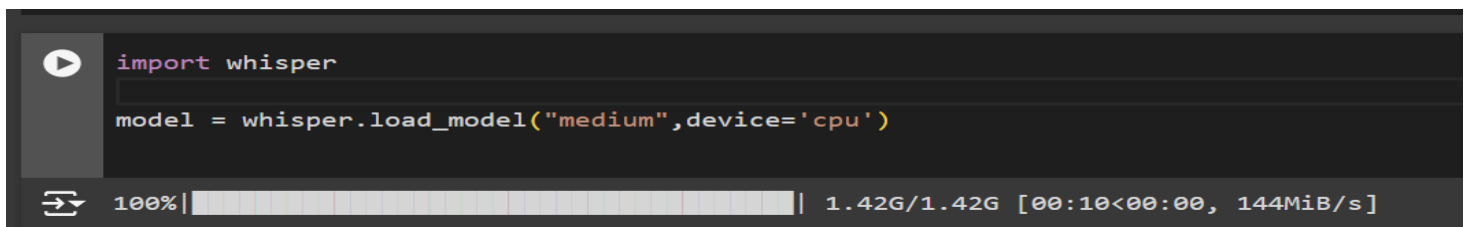
Explanation (Cell 1): OpenAI's Whisper is a sophisticated voice processing model intended for language identification, multilingual speech recognition, translation, and other uses. This cell installs the whisper library directly from its GitHub source using a Google notebook shell command (!). The notebook remains cleaner during installation when the -q parameter is used to suppress output.



```
!pip install transformers -q
!pip install torch -q
!pip install librosa -q
```

Explanation (Cell 2): The image shows a Jupyter notebook or Google Colab cell with three commands to install Python libraries:

- transformers: For natural language processing.
- torch: For deep learning with PyTorch.
- librosa: For audio and music analysis.



```
import whisper

model = whisper.load_model("medium", device='cpu')
```

100% | 1.42G/1.42G [00:10<00:00, 144MiB/s]

Explanation (Cell 3): The following code imports the whisper library and loads a pre-trained multilingual Whisper model named "medium" on the CPU device.

```

from google.colab import files

# Upload the audio file
uploaded = files.upload()

# Check the uploaded files
for filename in uploaded.keys():
    print(f"Uploaded file: {filename}")

```

Choose Files aud_1.mp3
 • aud_1.mp3(audio/mpeg) - 1348554 bytes, last modified: 8/2/2024 - 100% done
 Saving aud_1.mp3 to aud_1.mp3
 Uploaded file: aud_1.mp3

Explanation (Cell 4): This code allows you to upload audio files in Google Colab. It imports the file handling module, prompts for file uploads, and then prints the names of the uploaded files.

```

import librosa
audio_path = list(uploaded.keys())[0]
def load_audio_in_chunks(audio_path, chunk_duration=30):
    audio, sr = librosa.load(audio_path, sr=16000)
    total_duration = librosa.get_duration(y=audio, sr=sr)
    chunk_samples = int(chunk_duration * sr)
    for start_sample in range(0, len(audio), chunk_samples):
        yield audio[start_sample:start_sample + chunk_samples], sr
chunks = list(load_audio_in_chunks(audio_path))
print(f"Loaded {len(chunks)} chunks from the audio file.")

```

Loaded 2 chunks from the audio file.

Explanation (Cell 5):

- **Import Library:** The code imports the librosa library for audio processing.
- **Set Audio Path:** It gets the filename of the uploaded audio file using `audio_path = list(uploaded.keys())[0]`.
- **Define Function:** The `load_audio_in_chunks` function: Takes `audio_path` and `chunk_duration` (default is 30 seconds).
- **Generate Chunks:** The function is called, and the resulting chunks are stored in a list: `chunks = list(load_audio_in_chunks(audio_path))`.
- **Print Result:** It prints the number of chunks loaded from the audio file: `print(f'Loaded {len(chunks)} chunks from the audio file.')`

```

import whisper
model = whisper.load_model("medium", device="cpu")
transcriptions = []
for i, (audio_chunk, sr) in enumerate(chunks):
    result = model.transcribe(audio_chunk, fp16=False) #FP32 is used
    transcriptions.append(result["text"])
    print(f"Transcription of chunk {i+1}/{len(chunks)}: {result['text']}")
full_transcription = " ".join(transcriptions)
print("Full Transcription:", full_transcription)

```

Transcription of chunk 1/2: I'm gonna play a recording of people chanting, that is embarrassing, four times. After the fourth time, write down what y
 Transcription of chunk 2/2: Thoughts is infarct. You heard the same thing every single time. Our eyes and ears take in electrical signals which our b
 Full Transcription: I'm gonna play a recording of people chanting, that is embarrassing, four times. After the fourth time, write down what you hear.

Explanation (Cell 6):

- **Load Model:** Load the "medium" multilingual Whisper model for CPU with FP32 precision.
- **Initialize List:** Create an empty list for storing transcriptions.
- **Loop Through Chunks:** Process each audio chunk to get its transcription.
- **Store & Print:** Append each transcription to the list and print it.
- **Combine Transcriptions:** Merge all transcriptions into one string.
- **Print Result:** Display the full combined transcription.

```
from transformers import MarianMTModel, MarianTokenizer
language_models = {
    "de": "Helsinki-NLP/opus-mt-en-de", # German
    "es": "Helsinki-NLP/opus-mt-en-es", # Spanish
    "fr": "Helsinki-NLP/opus-mt-en-fr", # French
    "it": "Helsinki-NLP/opus-mt-en-it", # Italian
    "zh": "Helsinki-NLP/opus-mt-en-zh", # Chinese
    # Add more languages here as needed
}

print("Choose the target language code from the following options:")
for lang_code in language_models.keys():
    print(f"{lang_code} - {language_models[lang_code].split('-')[-1]}")
target_language = input("Enter the target language code: ")
if target_language not in language_models:
    print("Invalid language code. Please run the cell again and enter a valid code.")
else:
    model_name = language_models[target_language]
    tokenizer = MarianTokenizer.from_pretrained(model_name)
    translation_model = MarianMTModel.from_pretrained(model_name)
    full_transcription = " ".join(transcriptions)
    print("Full Transcription:", full_transcription)
    inputs = tokenizer(full_transcription, return_tensors="pt", padding=True)
    translated_tokens = translation_model.generate(**inputs)
    translated_text = tokenizer.decode(translated_tokens[0], skip_special_tokens=True)
    print("Translated Text:", translated_text)
```

This is the output of the above code where user need to select the language that he want to translate then it give the translated text.

```
Choose the target language code from the following options:
de - de
es - es
fr - fr
it - it
zh - zh
Enter the target language code: es
Full Transcription: I'm gonna play a recording of people chanting, that is embarrassing, four times. After the fourth time, write
Translated Text: Voy a tocar una grabación de gente cantando, eso es vergonzoso, cuatro veces. Después de la cuarta vez, escribir
```

Explanation (Cell 7):

- **Import Libraries:** Import the necessary classes from transformers.
- **Define Models:** Create a dictionary with language codes and corresponding translation model names.
- **Choose Language:** Print available language options and prompt the user to enter a language code.
- **Load Model:** If the code is valid, load the tokenizer and model for the chosen language.
- **Translate Text:** Tokenize a sample text, generate a translation, and decode it.
- **Print Result:** Display the translated text.

```
14s !pip install datasets
!pip install faiss-cpu

Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-packages (2.20.0)
```

Explanation (Cell 8):

- **Install datasets:** For dataset handling.
- **Install faiss-cpu:** For fast vector search and clustering on CPU.

```
from transformers import RagTokenizer, RagRetriever, RagSequenceForGeneration
tokenizer = RagTokenizer.from_pretrained("facebook/rag-token-nq")
retriever = RagRetriever.from_pretrained("facebook/rag-token-nq", index_name="exact", use_dummy_dataset=True)
rag_model = RagSequenceForGeneration.from_pretrained("facebook/rag-token-nq", retriever=retriever)
query = translated_text
input_ids = tokenizer(query, return_tensors="pt").input_ids
generated_ids = rag_model.generate(input_ids=input_ids)
response = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)
print("RAG Response:", response)
```

The class this function is called from is 'BartTokenizer'.
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may result in unexpected tokeniza
The tokenizer class you load from this checkpoint is 'RagTokenizer'.
The class this function is called from is 'BartTokenizerFast'.
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1249: UserWarning: Using the model-agnostic default 'max_length' (=20) to con
warnings.warn(
RAG Response: [" `` law of attraction ``"]

Explanation (Cell 9):

- **Import Libraries:** Load the necessary classes from transformers.
- **Initialize Components:** Set up the tokenizer, retriever, and RAG model using a pre-trained RAG model.
- **Prepare Input:** Tokenize the query text.
- **Generate Response:** Use the RAG model to generate a response.
- **Decode & Print:** Decode the generated response and print it.

DEMONSTRATION(the input I have taken):

The original recording lasts 55 seconds. The audio was sliced and diced into little segments of 30 seconds each. Each part was rendered into English individually to get word-for-accuracy and completeness. Once the English translations were obtained, the texts were then turned into Spanish according to user input and you get the final Spanish output.



ORIGINAL TEXT OF AUDIO IN ENGLISH:

I'm gonna play a recording of people chanting, that is embarrassing, four times. After the fourth time, write down what you hear. Thoughts is embarrassing! Thoughts is embarrassing! Thoughts is embarrassing! Thoughts is embarrassing! Interesting, almost everyone here heard, that is embarrassing. Listen again and watch the screen. Thoughts is embarrassing! Thoughts is embarrassing! Thoughts is embarrassing! Thoughts is infarct. You heard the same thing every single time. Our eyes and ears take in electrical signals which our brains interpret based on our expectations. We don't see reality. We see our reality.

Translated Text in Spanish:

Voy a tocar una grabación de gente cantando, eso es vergonzoso, cuatro veces. Después de la cuarta vez, escribir lo que escuchas. ¡Los pensamientos es vergonzoso! ¡Los pensamientos es embarazoso! ¡Los pensamientos es embarazoso! ¡Los pensamientos es vergonzoso! Interesante, casi todos aquí escucharon, eso es vergonzoso. Escucha de nuevo y mira la pantalla. ¡Los pensamientos es vergonzoso! ¡Los pensamientos es vergonzoso! ¡Los pensamientos es vergonzoso! ¡Los pensamientos son infartos. Escuchaste la misma cosa cada vez. Nuestros ojos y oídos toman señales eléctricas que nuestros cerebros interpretan en base a nuestras expectativas. No vemos la realidad. Vemos nuestra realidad.

Enter the target language code: es

Full Transcription: I'm gonna play a recording of people chanting, that is embarrassing, four times. After the fourth time, write down what you hear.

Translated Text: Voy a tocar una grabación de gente cantando, eso es vergonzoso, cuatro veces. Después de la cuarta vez, escribir lo que escuchas. ¡Lo.

References:

<https://github.com/nikkibommu/Building-a-Multilingual-Speech-Recognition-Model-for-RAG-Without-Training>