# Report

**File Building a Multilingual Speech Recognition Model for RAG Without Training.**

Submitted by: Sakshi Upadhyay

Submitted to: TensorGo

Date: 8th August, 2024

# Index

# Introduction

Building a Multilingual Speech Recognition Model for Retrieval-Augmented Generation (RAG) without training involves utilizing pre-existing, pre-trained models that support multiple languages. This approach bypasses the need for extensive data collection and model training by leveraging advanced models like Whisper, which have been trained on diverse datasets. By integrating these models into the RAG framework, we can efficiently handle multilingual speech-to-text tasks with minimal additional work.

# Objective

The objective of this project is to develop a robust, scalable multilingual speech recognition system that can be seamlessly integrated into RAG pipelines without the need for retraining. The goal is to enable accurate speech-to-text conversion across various languages, improving the accessibility and effectiveness of RAG models in processing and understanding multilingual inputs.

# Background

Recent advancements in speech recognition and natural language processing (NLP) have led to the creation of powerful pre-trained models, such as Whisper, that support multiple languages. These models have been trained on vast amounts of data, enabling them to perform well in diverse linguistic contexts. In parallel, RAG frameworks have emerged as a leading approach to enhance generative AI by incorporating external knowledge through retrieval mechanisms, making them ideal for tasks requiring understanding and processing of multilingual speech inputs.

# Methodology

The methodology involves selecting a suitable pre-trained multilingual speech recognition model, such as Whisper, and integrating it into a RAG pipeline. The process includes fine-tuning the model's parameters to optimize performance in the RAG context, ensuring that it can handle diverse language inputs. Additionally, the system will be tested and evaluated on a range of multilingual datasets to validate its accuracy and effectiveness in real-world scenarios.

# Code explanation

## 1. Install Required Packages

!pip install -U openai-whisper

!pip install sentence_transformers

! pip install gradio==3.50.2

openai-whisper: To record spoken words.
Gradio: To build an online user interface.
sentence_transformers: To manage the retrieval of documents by means of embeddings.

## 2. Import Libraries and Load Models

import whisper

from transformers import RagTokenizer, RagRetriever, RagSequenceForGeneration

from sentence_transformers import SentenceTransformer, util

from transformers import MarianMTModel, MarianTokenizer, pipeline

import torch

import gradio as gr

import whisper-: Imports the Whisper speech recognition model from OpenAI.

from transformers import RagTokenizer, RagRetriever, RagSequenceForGeneration-: Imports components for a Retrieval-Augmented Generation (RAG) model from the Hugging Face `transformers` library.

from sentence_transformers import SentenceTransformer, util-: Imports the SentenceTransformer model and utility functions for embedding and similarity tasks.

from transformers import MarianMTModel, MarianTokenizer, pipeline-: Imports the MarianMT translation model and tokenizer for language translation tasks.

import torch-: Imports PyTorch, a deep learning framework used for model operations.

import gradio as gr-: Imports Gradio, a library for creating web-based interfaces to interact with machine learning models.

## 3. Create a Dummy Document Store and Encode Document Embeddings

model_name = "facebook/rag-sequence-nq" tokenizer = RagTokenizer.from_pretrained(model_name) model = RagSequenceForGeneration.from_pretrained(model_name)

# Import the Whisper multilingual model.

whisperModel = whisper.load_model("base")

#Import Sentence Transformer in order to extract

retrieverModel = SentenceTransformer('all-MiniLM-L6-v2')

# Specify a document structure akin to RAG.

documents = {

"doc1": {"title": "Artificial Intelligence and Machine Learning", "content": "This document describes the fundamentals of artificial intelligence and machine learning."},

"doc2": {"title": "Deep Learning and Neural Networks", "content": "This document explores the concepts of deep learning and neural networks."},

"doc3": {"title": "Impact of AI on Different Industries", "content": "This document discusses the impact of AI on various sectors like healthcare, finance, and more."},

"doc4": {"title": "Future of Technology", "content": "This document highlights advancements in AI, quantum computing, and other fields shaping the future of technology."},

"doc5": {"title": "The Promise of AI", "content": [

"The future of AI holds immense potential, marked by rapid progress and transformative capabilities across various industries.",

"AI is expected to revolutionize healthcare, finance, transportation, and education by enabling more efficient and accurate decision-making processes.",

"In healthcare, AI can support early diagnosis, personalized treatment plans, and even robotic surgeries.",

"The finance sector can leverage AI-driven algorithms for fraud detection and optimal trading strategies.",

"The transportation sector is likely to see widespread adoption of autonomous vehicles, improving safety and reducing congestion.",

"Education can benefit from personalized learning experiences tailored to individual student needs."

]

}

}

model_name = "facebook/rag-sequence-nq-: Specifies the name of the pre-trained RAG model, which is designed for natural question answering (NQ).

tokenizer = RagTokenizer.from_pretrained(model_name)-: Loads the tokenizer associated with the `rag-sequence-nq` model. The tokenizer converts input text into tokens that the model can process.

model = RagSequenceForGeneration.from_pretrained(model_name)-: Loads the pre-trained RAG sequence model. This model is used for generating text in response to input queries by retrieving relevant information and generating answers.

The `documents` dictionary simulates a RAG-like structure, containing multiple documents with titles and content related to AI topics. Each document's content can be a single string or a list of strings. This structure is used for retrieving relevant information in RAG models.

## 4. Define Functions for Speech Transcription, Language Detection, and Translation

```
# Condense the content of the document for encoding.


allDocumentTexts = []

for key, value in documents.items():
    if isinstance(value, list):
        allDocumentTexts.extend(value)
    else:
        allDocumentTexts.append(value)


# Use the retriever model to encrypt the documents.


documentEmbeddings = retrieverModel.encode(allDocumentTexts, convert_to_tensor=True)


def transcribeSpeech(filePath):
    """Transcribes speech using Multilingual Whisper"""
```

```
    result = whisperModel.transcribe(filePath)
    return result["text"]

def detectLanguage(audioPath):
    """Detects the language of the audio using Multilingual Whisper"""
    audio = whisper.load_audio(audioPath)
    audio = whisper.pad_or_trim(audio)
    mel = whisper.log_mel_spectrogram(audio).to(whisperModel.device)
    _, probs = whisperModel.detect_language(mel)
    detectedLanguageCode = max(probs, key=probs.get)

    languageMapping = {
        'en': 'English', 'es': 'Spanish', 'fr': 'French', 'de': 'German',
        'hi': 'Hindi', 'ja': 'Japanese', 'ru': 'Russian', 'ar': 'Arabic',
        'te': 'Telugu', 'zh': 'Chinese', 'pt': 'Portuguese'
    }

    return languageMapping.get(detectedLanguageCode, detectedLanguageCode).capitalize()

def loadTranslationModel(sourceLanguage, targetLanguage):
    """Loads translation model and tokenizer based on languages"""
    modelName = {
        ("English", "Hindi"): "Helsinki-NLP/opus-mt-en-hi",
        ("English", "Spanish"): "Helsinki-NLP/opus-mt-en-es",
        ("English", "Japanese"): "Helsinki-NLP/opus-mt-en-jap",
        ("English", "German"): "Helsinki-NLP/opus-mt-en-de",
        ("English", "Russian"): "Helsinki-NLP/opus-mt-en-ru",
        ("English", "Arabic"): "Helsinki-NLP/opus-mt-en-ar",
        ("English", "Telugu"): "Helsinki-NLP/opus-mt-en-te",
```

```
        ("English", "French"): "Helsinki-NLP/opus-mt-en-fr",
        ("English", "Italian"): "Helsinki-NLP/opus-mt-en-it",
        ("English", "Chinese"): "Helsinki-NLP/opus-mt-xx-zh",
        ("Hindi", "English"): "Helsinki-NLP/opus-mt-hi-en",
        ("Spanish", "English"): "Helsinki-NLP/opus-mt-es-en",
        ("Japanese", "English"): "Helsinki-NLP/opus-mt-jap-en",
        ("German", "English"): "Helsinki-NLP/opus-mt-de-en",
        ("Russian", "English"): "Helsinki-NLP/opus-mt-ru-en",
        ("Arabic", "English"): "Helsinki-NLP/opus-mt-ar-en",
        ("Telugu", "English"): "Helsinki-NLP/opus-mt-te-en",
        ("French", "English"): "Helsinki-NLP/opus-mt-fr-en",
        ("Italian", "English"): "Helsinki-NLP/opus-mt-it-en",
        ("Chinese", "English"): "Helsinki-NLP/opus-mt-zh-en"
    }

    if (sourceLanguage, targetLanguage) not in modelName:
        raise ValueError(f"Translation model for {sourceLanguage} to {targetLanguage} not
available.")

    translationModel = MarianMTModel.from_pretrained(modelName[(sourceLanguage,
targetLanguage)])
    translationTokenizer = MarianTokenizer.from_pretrained(modelName[(sourceLanguage,
targetLanguage)])
    return translationModel, translationTokenizer

def detectLanguage(audioPath):
    """Detects the language of the audio using Multilingual Whisper"""
    audio = whisper.load_audio(audioPath)
    audio = whisper.pad_or_trim(audio)
    mel = whisper.log_mel_spectrogram(audio).to(whisperModel.device)
```

```
    _, probs = whisperModel.detect_language(mel)
    detectedLanguageCode = max(probs, key=probs.get)

    languageMapping = {
        'en': 'English', 'es': 'Spanish', 'fr': 'French', 'de': 'German',
        'hi': 'Hindi', 'ja': 'Japanese', 'ru': 'Russian', 'ar': 'Arabic',
        'te': 'Telugu', 'zh': 'Chinese', 'pt': 'Portuguese'
    }

    return languageMapping.get(detectedLanguageCode, detectedLanguageCode).capitalize()

def loadTranslationModel(sourceLanguage, targetLanguage):
    """Loads translation model and tokenizer based on languages"""
    modelName = {
        ("English", "Hindi"): "Helsinki-NLP/opus-mt-en-hi",
        ("English", "Spanish"): "Helsinki-NLP/opus-mt-en-es",
        ("English", "Japanese"): "Helsinki-NLP/opus-mt-en-jap",
        ("English", "German"): "Helsinki-NLP/opus-mt-en-de",
        ("English", "Russian"): "Helsinki-NLP/opus-mt-en-ru",
        ("English", "Arabic"): "Helsinki-NLP/opus-mt-en-ar",
        ("English", "Telugu"): "Helsinki-NLP/opus-mt-en-te",
        ("English", "French"): "Helsinki-NLP/opus-mt-en-fr",
        ("English", "Italian"): "Helsinki-NLP/opus-mt-en-it",
        ("English", "Chinese"): "Helsinki-NLP/opus-mt-xx-zh",
        ("Hindi", "English"): "Helsinki-NLP/opus-mt-hi-en",
        ("Spanish", "English"): "Helsinki-NLP/opus-mt-es-en",
        ("Japanese", "English"): "Helsinki-NLP/opus-mt-jap-en",
        ("German", "English"): "Helsinki-NLP/opus-mt-de-en",
        ("Russian", "English"): "Helsinki-NLP/opus-mt-ru-en",
```

```
        ("Arabic", "English"): "Helsinki-NLP/opus-mt-ar-en",
        ("Telugu", "English"): "Helsinki-NLP/opus-mt-te-en",
        ("French", "English"): "Helsinki-NLP/opus-mt-fr-en",
        ("Italian", "English"): "Helsinki-NLP/opus-mt-it-en",
        ("Chinese", "English"): "Helsinki-NLP/opus-mt-zh-en"
    }

    if (sourceLanguage, targetLanguage) not in modelName:
        raise ValueError(f"Translation model for {sourceLanguage} to {targetLanguage} not available.")

    translationModel = MarianMTModel.from_pretrained(modelName[(sourceLanguage, targetLanguage)])
    translationTokenizer = MarianTokenizer.from_pretrained(modelName[(sourceLanguage, targetLanguage)])
    return translationModel, translationTokenizer
```

transcribe_speech: Uses the Whisper model to transcribe the audio file.
detect_language: Determines the audio file's language.
load_translation_model: This function loads the tokenizer and relevant translation model.
translate_text: Uses the loaded model to translate the transcribed text.

## 5. Define Function for Document Retrieval

```
def retrieveDocument(query):
    """Retrieves a document based on the query"""
    queryEmbedding = retrieverModel.encode(query, convert_to_tensor=True)
    scores = util.pytorch_cos_sim(queryEmbedding, documentEmbeddings)[0]
    topScoreIdx = scores.argmax().item()
    return list(documents.values())[topScoreIdx]

# Pipeline for load summarization
```

summarizer = pipeline("summarization", model="facebook/bart-large-cnn")

def summarizeText(text):

"""Summarizes the text"""

summary = summarizer(text, max_length=50, min_length=25, do_sample=False)

return summary[0]['summary_text']

retrieve_document: This function uses sentence embeddings to return the document that best fits the query.

## 6. Define Main Function to Process Audio and Generate Outputs

def processAudio(audio, targetLanguage):

"""Processes the audio file and returns transcriptions, translations, and summaries"""

transcription = transcribeSpeech(audio)

detectedLanguage = detectLanguage(audio)

# If the transcription isn't already in English, translate it there.

if detectedLanguage != "English":

translationModel, translationTokenizer = loadTranslationModel(detectedLanguage, "English")

transcriptionInEnglish = translateText(transcription, translationModel, translationTokenizer)

else:

transcriptionInEnglish = transcription

# If the target language is not English, translate the English transcription to the other language.

if targetLanguage != "English":

```
        translationModel, translationTokenizer = loadTranslationModel("English", targetLanguage)
        translatedText = translateText(transcriptionInEnglish, translationModel, translationTokenizer)
    else:
        translatedText = transcriptionInEnglish

# Get the document using the text transcription as a guide.

    retrievedDocument = retrieveDocument(transcriptionInEnglish)
    summarizedText = summarizeText(transcriptionInEnglish)
    return transcription, detectedLanguage, translatedText, retrievedDocument, summarizedText
```

process_audio: This function combines language detection, transcription, translation, and document retrieval in order to process audio and produce outputs.

## 7. Create and Launch Gradio Interface

```
# Develop the interface for Gradio.
iface = gr.Interface(
    fn=processAudio,
    inputs=[
        gr.Audio(source="upload", type="filepath"),
        gr.Dropdown(["Hindi", "Spanish", "Japanese", "German", "Russian", "Arabic", "French", "Italian", "Chinese", "English"], label="Target Language")
    ],
    outputs=[
        gr.Textbox(label="Transcription"),
        gr.Textbox(label="Detected Language"),
        gr.Textbox(label="Translation"),
        gr.Textbox(label="Retrieved Document"),
```

```
        gr.Textbox(label="Summarized Text")
    ],
    title="Multilingual Speech Recognition, Translation, Document Retrieval, and Summarization",
    description="Upload an audio file in any language, select a target language to get the transcription, translation, retrieve a document based on the transcription, and get a summary of the transcription."
)

# Start the Gradio user interface.

iface.launch()
```

Gradio Interface: Launches and defines the Gradio interface's input and output components.

# Result

❖ German to English:

- Transcription:
Artificial intelligence emerges as a contemporary muse in a world where creativity and imagination coexist, shaping a future in which machines not only mimic but also surpass human thought. Imagine an AI painting a canvas with the colors of neural networks and deep learning, creating data symphonies that whisper the secrets of the universe. It's a dance of creativity and algorithms, a coming together of art and science, where AI takes on the role of the silent progressor, bringing the melody of endless possibilities to the beat of tomorrow.

- Translation:

In a world where creativity and imagination are stifled, artificial intelligence emerges as a contemporary Muse, modeling a future in which machines not only mimic human thought processes but also transcend them.
Imagine a line in which the mysteries of the universe are revealed by neural networks and the color tones of deep learning. It's a dance of algorithms and creativity, a fusion of art and science, where KI becomes the silent composer of the future, harmonizing the rhythm of death with the melody of limitless possibilities.

English to Spanish:

- Transcription:

Artificial intelligence emerges as a contemporary muse in a world where creativity and imagination coexist, shaping a future in which machines not only mimic but also surpass human thought. Imagine an AI painting a canvas with the colors of neural networks and deep learning, creating data symphonies that whisper the secrets of the universe. It's a dance of creativity and algorithms, a coming together of art and science, where AI takes on the role of the silent progressor, bringing the melody of endless possibilities to the beat of tomorrow.

- Translation:

In a world where creativity and innovation coexist, artificial intelligence emerges as a contemporary goddess, threatening a future in which machines not only mimic but also transcend human thought processes.
Imagine a world in which artificial intelligence (AI) paints with the brushstrokes of deep learning and neural networks, creating data symphonies that hold the secrets of the cosmos. It's a dance of algorithms and creativity, a coming together of science and art, where AI becomes the silent composer of progress, harmonizing the pace of the modern world with the limitless possibilities.

# Conclusion

The deployed system shows that it is capable of efficiently translating and transcribing speech in a number of languages. We were able to achieve high accuracy without the need for further training by utilizing pre-trained models.

# Future work

➢ Potential enhancements could involve: • Including additional language support.
• Improving the user interface to improve the experience.
• Optimizing the model for faster performance.

# References

OpenAI Whisper: GitHub Repository

▪ Hugging Face Marian MT: Documentation

▪ Gradio: Documentatio