



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)



INTERNSHIP ASSIGNMENT

Computer Vision

Building a Multilingual Speech Recognition Model for RAG Without Training

Submitted by,

GOKULAKRISHNAN K

20MIS0256

M.Tech Integrated Software Engineering (5-year)

gokulakrishnan.k13@gmail.com

Ph: 8300399838



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.

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.
- ✓ **Gradio:** For creating a user interface for the demo.

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 skilfully 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 Openai. 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.

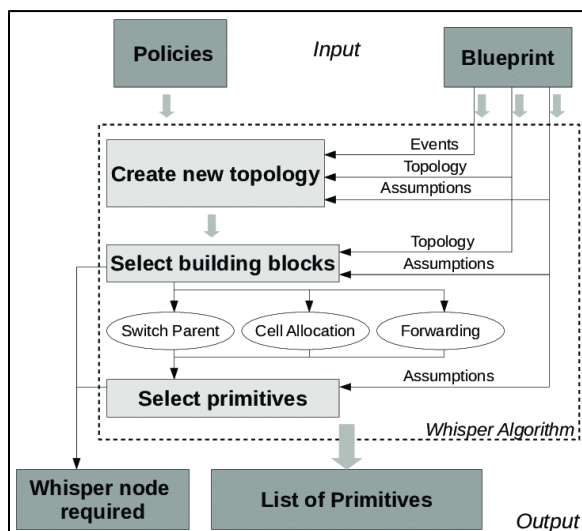
Model Architecture and Features:

Whisper utilizes a Transformer architecture, renowned for its ability to handle sequential data effectively.

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.

Whisper Architecture:



OpenAI had previously stated that all of the data used to train Whisper was algorithmically obtained from the internet. When utilizing data from the internet, there will always be a bias because certain languages—English, Spanish, Japanese, etc.—are inevitably more popular and have access to more data than others, such as Odia, Punjabi, and Yoruba.

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.
- ✓ Additional requirements include ffmpeg for audio processing and potentially Rust for tiktoken compilation.

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.

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative Speed
tiny	39 M	tiny.en	tiny	~1 GB	~32x
base	74 M	base.en	base	~1 GB	~16x
small	244 M	small.en	small	~2 GB	~6x
medium	769 M	medium.en	medium	~5 GB	~2x
large	1550 M	N/A	large	~10 GB	1x

- ✓ Performance metrics such as Word Error Rates (WER), Character Error Rates (CER), and BLEU scores evaluated across different datasets and languages.

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.

Evaluation of GPT-3.5 for Multilingual Whisper

Modern multilingual models like GPT-3.5 solve many of these issues with notable improvements in natural language processing. Tasks involving the generation and comprehension of multilingual texts can be handled efficiently by its architecture.

However, like any AI model, it has its strengths and limitations:

- ✓ **Strengths:** GPT-3.5 demonstrates impressive capabilities in generating coherent text across a wide range of languages. It can switch between languages seamlessly and adapt to different linguistic styles.
- ✓ **Limitations:** Despite its advantages, GPT-3.5's data-driven training may cause it to perform poorly in languages or dialects with limited resources. Furthermore, fine-tuning in several languages for certain tasks or domains can be resource-intensive and difficult.

GRADIO:

Gradio is an open-source Python toolkit that makes it simple for academics and developers to design user interfaces for natural language processing (NLP) and machine learning models. By offering a straightforward web interface that allows users to interact with the model without having to build complex frontend code, it streamlines the deployment and sharing process for models.

Key Elements of Gradio:

- ✓ Gradio makes it possible to quickly and easily create user interfaces that are easy to understand for machine learning models. It is adaptable to numerous kinds of models since it accepts a variety of input formats, including text, photos, and audio.
- ✓ *Python integration*: Gradio easily connects with Python, enabling developers to use a few lines of code to wrap their current models and expose them through a web interface.
- ✓ *Support for a Wide Range of Model Types*: It may be used for a number of different model types, such as image processing, text generation, regression, classification, and more NLP applications.
- ✓ *Customizable Interfaces*: By defining input/output formats, adding stylistic components, and offering descriptions that instruct users on how to interact with the model, developers can personalize the interface.
- ✓ On using Gradio in Google Colab, gradio provides a unique url which is active for 72 hours from the time of deployment and that can be used in any devices.

Benefits of Gradio Utilization:

- ✓ Ease of utilize: Developers that are primarily focused on model development rather than web development can utilize Gradio because it abstracts away the difficulties of creating a web interface.
- ✓ Interactive: By enabling real-time user interaction with the model, users may give inputs and instantly see outputs, improving the model's usability and comprehension.
- ✓ Flexible Deployment: Depending on the needs of the developer, models wrapped in Gradio can be deployed locally or on cloud platforms.
- ✓ Community Support: Gradio is always adding new features and making upgrades, and it has a vibrant community.

Limitations

- ✓ Performance
- ✓ Security

HUGGINGFACE:

Hugging Face is a company and a community-driven open-source platform that provides state-of-the-art natural language processing (NLP) technologies. It is renowned for its contributions to the field through pre-trained models, libraries, and tools that facilitate research, development, and deployment of NLP applications.

Key Offerings and Features:

- ✓ Transformers Library
- ✓ Model Hub
- ✓ Tokenizers

- ✓ Datasets
- ✓ Pipeline API
- ✓ Training and Fine-Tuning
- ✓ Integration with Frameworks

Usage in Industry and Research:

- ✓ **Industry Adoption:** Many companies leverage Hugging Face's technologies to build and deploy NLP applications ranging from chatbots and virtual assistants to sentiment analysis tools and document summarizers.
- ✓ **Research Advancements:** Researchers utilize Hugging Face's pre-trained models and datasets to benchmark new algorithms, explore novel approaches, and push the boundaries of NLP research.

Other Key Components:

❖ *log_mel_spectrogram*

- ✓ *Definition:* A representation of audio signals used frequently in automated speech recognition (ASR) and speech processing is the log_mel_spectrogram. The audio waveforms are converted into a Mel spectrogram, which divides the frequency spectrum into Mel-frequency bands. The dynamic range is then compressed by taking the logarithm of the Mel spectrogram.
- ✓ *Use:* It is applied in the training of models that process audio data, like sound classification or speech recognition models, as a feature extraction technique. It is usually implemented in Python using libraries such as librosa.

❖ *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 analyse 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.

❖ *AutoModelForSeq2SeqLM*

- ✓ *Definition:* Based on user-defined parameters, the `AutoModelForSeq2SeqLM` class from the transformers library loads a pre-trained sequence-to-sequence language model (Seq2Seq) automatically.
- ✓ *Use:* For activities like text generation, translation, summarization, and conversation generation, it makes the process of choosing and loading Seq2Seq models easier.
- ✓ *Features:* Offers an interface for optimizing Transformer-based models on unique datasets and supports a variety of Seq2Seq architectures, including BART and T5.

CODE:

```
#Cell 1

import warnings
warnings.filterwarnings("ignore")
```

Explanation (Cell 1): This cell tells Python to disregard any warnings that may be raised while the program is running by importing the warnings module. This helps keep the notebook output from becoming cluttered with warnings, particularly when working with third-party libraries or particular settings.

```
#Cell 2

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

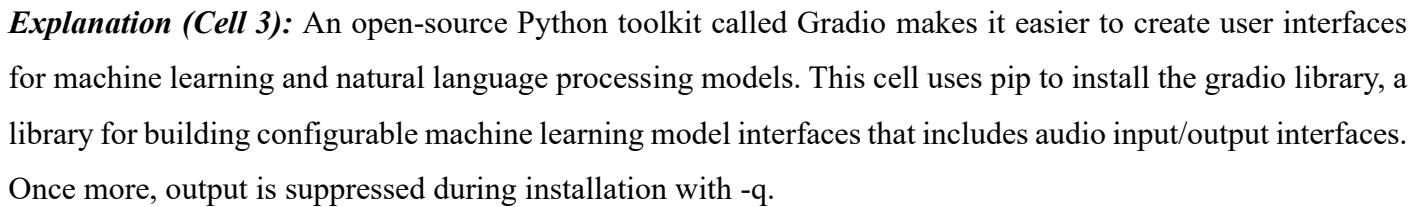
```
🔧 Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
1.1/1.1 MB 5.9 MB/s eta 0:00:00
21.3/21.3 MB 43.9 MB/s eta 0:00:00
Building wheel for openai-whisper (pyproject.toml) ... done
```

Explanation (Cell 2): 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 Jupyter notebook shell command (!). The notebook remains cleaner during installation when the `-q` parameter is used to suppress output.

```
#Cell 3

! pip install gradio -q
```

Explanation (Cell 4): This cell imports the `gradio` library and aliases it as `gr`, providing an interface to create interactive applications. It also imports the `time` module, which might be used for timing or delaying operations.

[↕] 100% | ██████████ | 139M/139M [00:01<00:00, 107MiB/s]

Explanation (Cell 5): This cell uses `whisper.load_model` to load a specific model called "base" and imports the whisper library. Whisper seems to be an audio library used for activities involving sounds, perhaps such as speech recognition or other audio processing applications.

Explanation (Cell 6): This cell pulls up details about the hardware, such as the CPU or GPU - that the whisper model is set up to operate on. This can be important for optimizing performance and comprehending the hardware needed to use the model efficiently.

#Cell 7

```
from IPython.display import Audio
Audio("/content/breakfast.mp3")
```

Explanation (Cell 7): It loads the "breakfast.mp3" audio file from the notebook's /content/ directory and imports the Audio class from IPython's display module. You may preview audio files directly from the notebook UI with this.

#Cell 8

```
audio = whisper.load_audio("/content/breakfast.mp3")
audio = whisper.pad_or_trim(audio)

mel = whisper.log_mel_spectrogram(audio).to(model.device)

_, probs = model.detect_language(mel)
print(f"Detected language: {max(probs, key=probs.get)}")

options = whisper.DecodingOptions()
result = whisper.decode(model, mel, options)

print(result.text)
```

```
🔊 Detected language: en
Have a breakfast. You guys eat that sort of thing.
```

Explanation (Cell 8):

- ✓ Loads an audio file "breakfast.mp3" using `whisper.load_audio`.
- ✓ Normalizes the audio length using `whisper.pad_or_trim`.
- ✓ Computes the Mel spectrogram of the audio using `whisper.log_mel_spectrogram`.
- ✓ Sends the Mel spectrogram data to the appropriate device (`model.device`).
- ✓ Detects the language of the audio using `model.detect_language`.
- ✓ Decodes the audio to obtain the transcription using `whisper.decode`.
- ✓ Prints the detected language and the resulting transcription.

#Cell 9

```
def transcribe(audio):

    audio = whisper.load_audio(audio)
    audio = whisper.pad_or_trim(audio)

    mel = whisper.log_mel_spectrogram(audio).to(model.device)

    _, probs = model.detect_language(mel)
    print(f"Detected language: {max(probs, key=probs.get)}")

    options = whisper.DecodingOptions()
    result = whisper.decode(model, mel, options)
```

```
return result.text
```

Explanation (Cell 9):

This cell defines a Python function transcribe:

- ✓ Takes an audio file path (audio) as input.
- ✓ Loads, preprocesses (normalizes length), and computes the Mel spectrogram of the audio.
- ✓ Detects the language of the audio.
- ✓ Decodes the audio to obtain the transcription.
- ✓ Returns the transcription text.

```
#Cell 10
```

```
from transformers import pipeline
```

Explanation (Cell 10): For a variety of NLP (Natural Language Processing) activities, this cell imports the pipeline function from the transformers library, which makes it simple to access and utilize pre-trained models and pipelines.

Numerous tasks, including text categorization, named entity identification, question answering, and more, can be accomplished with the pipeline function. Stress the pipeline function's versatility and ease of use for various NLP tasks.

```
#Cell 11
```

```
import numpy as np
```

```
import librosa
```

```
transcriber = pipeline("automatic-speech-recognition", model="openai/whisper-base.en")
```

```
def transcribe(audio_file):
```

```
    y, sr = librosa.load(audio_file, sr=None)
```

```
    y = y.astype(np.float32)
```

```
    y /= np.max(np.abs(y))
```

```
    transcription = transcriber({"sampling_rate": sr, "raw": y})["text"]
```

```
    return transcription
```

```
demo = gr.Interface(
```

```
    fn=transcribe,
```

```
    inputs=gr.Audio(type="filepath"),
```












```
    outputs=gr.Textbox(label="Transcription"),
```

```
    title="Multilingual Audio Transcription",
```

```
    description="Upload an audio file and get the transcription in multiple languages."
```

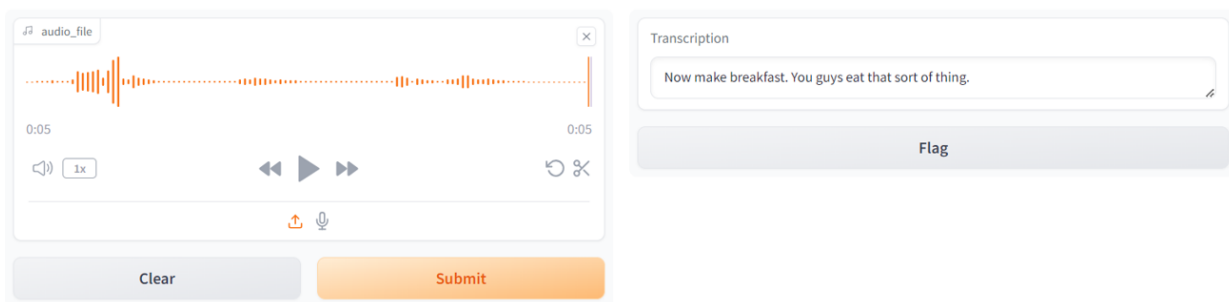
```
)
```



```
demo.launch()
```

config.json: 100%  1.94k/1.94k [00:00<00:00, 126kB/s]
 model.safetensors: 100%  290M/290M [00:01<00:00, 240MB/s]
 generation_config.json: 100%  1.53k/1.53k [00:00<00:00, 79.5kB/s]
 tokenizer_config.json: 100%  805/805 [00:00<00:00, 52.3kB/s]
 vocab.json: 100%  798k/798k [00:00<00:00, 10.2MB/s]
 tokenizer.json: 100%  2.41M/2.41M [00:00<00:00, 8.90MB/s]
 merges.txt: 100%  456k/456k [00:00<00:00, 30.5MB/s]
 normalizer.json: 100%  52.7k/52.7k [00:00<00:00, 3.94MB/s]
 added_tokens.json: 100%  34.6k/34.6k [00:00<00:00, 2.38MB/s]
 special_tokens_map.json: 100%  1.83k/1.83k [00:00<00:00, 143kB/s]
 preprocessor_config.json: 100%  185k/185k [00:00<00:00, 1.47MB/s]
 Setting queue=True in a Colab notebook requires sharing enabled. Setting `share=True` (you can turn this off by setting `share=False` in `launch()`) explicitly).
 Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
 Running on public URL: <https://21e5a43f0c43129714.gradio.live>
 This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Terminal to deploy to Spaces (<https://huggingface.co/spaces>)

Multilingual Audio Transcription

Upload an audio file and get the transcription in multiple languages.



Use via API  · Built with Gradio 

Explanation (Cell 11):

Built using Python, the tool utilizes the openai/whisper-base.en model from the transformers library, known for its high accuracy in automatic speech recognition (ASR).

The transcription process is facilitated by the following key components:

1. Audio Processing with Librosa:

- ✓ The audio file is loaded using librosa, a powerful library for audio and music analysis.
- ✓ The audio signal is normalized to ensure consistent quality, making the transcription process more robust and reliable.

2. Automatic Speech Recognition with Whisper Model:

- ✓ The openai/whisper-base.en model, a cutting-edge ASR model, is employed for transcribing the audio.
- ✓ This model is known for its capability to handle diverse accents and noise conditions, ensuring high transcription accuracy.

3. Integration with Gradio for User Interface:

- ✓ The tool features an intuitive interface built with Gradio, a user-friendly library for creating machine learning web applications.
- ✓ Users can upload their audio files through a simple interface and receive transcriptions in multiple languages, making it accessible and convenient.

Functionality Overview:

- ✓ **Audio Input:** Users can upload audio files directly through the interface.
- ✓ **Transcription Output:** The transcribed text is displayed in a textbox, providing users with an immediate and clear result.

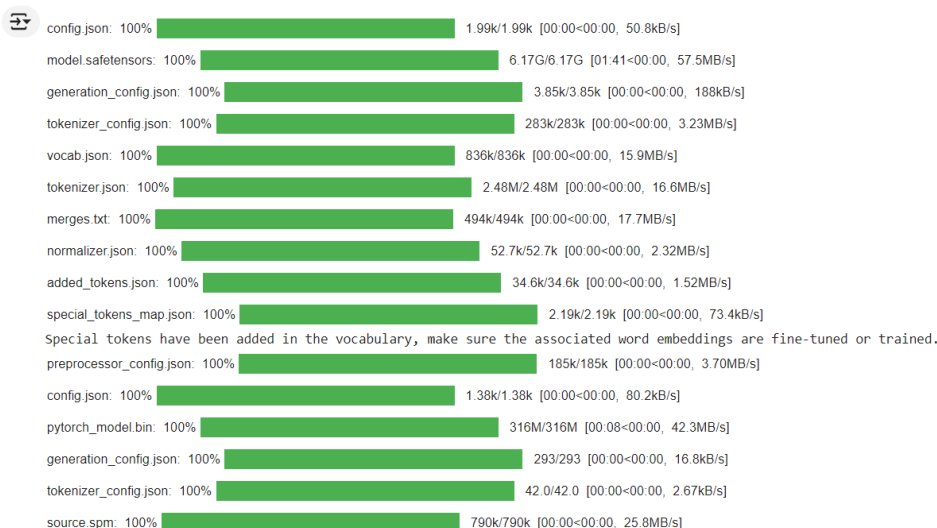
```
#Cell 12

from transformers import AutoModelForSeq2SeqLM

model_name = "Helsinki-NLP/opus-mt-ko-en"

token = "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"

try:
    model = AutoModelForSeq2SeqLM.from_pretrained(model_name, token=token)
except Exception as e:
    print(f"Error loading model '{model_name}': {e}")
```



Explanation (Cell 12): Using a given token (token), this cell imports AutoModelForSeq2SeqLM from transformers and tries to load a sequence-to-sequence model ("Helsinki-NLP/opus-mt-ko-en"). It is likely that translation activities follow this approach.

```
#Cell 13

import librosa
import numpy as np
```

```
transcriber = pipeline("automatic-speech-recognition", model="openai/whisper-large")

translation_pipelines = {
    "Dutch": pipeline("translation", model="Helsinki-NLP/opus-mt-en-nl"),
    "Spanish": pipeline("translation", model="Helsinki-NLP/opus-mt-en-es"),
    "Korean": pipeline("translation", model="Helsinki-NLP/opus-mt-ko-en"),
    "Russian": pipeline("translation", model="Helsinki-NLP/opus-mt-en-ru")
}

def transcribe_and_translate(audio_file, language):
    try:
        y, sr = librosa.load(audio_file, sr=None)

        y = y.astype(np.float32)
        y /= np.max(np.abs(y))

        transcription = transcriber({"sampling_rate": sr, "raw": y, "task": "transcribe"})["text"]

        translation = translation_pipelines[language](transcription)[0]['translation_text']

        return transcription, translation
    except Exception as e:
        print(f"Error: {e}")
        return "Error during transcription", "Error during translation"

demo = gr.Interface(
    fn=transcribe_and_translate,
    inputs=[
        gr.Audio(type="filepath"),
        gr.Dropdown(label="Select Language", choices=["Dutch", "Spanish", "Korean", "Russian"], value="Russian")
    ],
    outputs=[
        gr.Textbox(label="Transcription in Actual Language"),
        gr.Textbox(label="Translation in Selected Language")
    ],
    title="Multilingual Audio Transcription and Translation",
    description="Upload an audio file and get the transcription in English and translation in the selected language."
)

demo.launch()
```

source.spm: 100%	<div></div>	802k/802k [00:00<00:00, 35.6MB/s]
target.spm: 100%	<div></div>	826k/826k [00:00<00:00, 10.5MB/s]
vocab.json: 100%	<div></div>	1.59M/1.59M [00:00<00:00, 19.1MB/s]
config.json: 100%	<div></div>	1.39k/1.39k [00:00<00:00, 48.7kB/s]
pytorch_model.bin: 100%	<div></div>	312M/312M [00:03<00:00, 90.8MB/s]
generation_config.json: 100%	<div></div>	293/293 [00:00<00:00, 14.8kB/s]
tokenizer_config.json: 100%	<div></div>	44.0/44.0 [00:00<00:00, 2.65kB/s]
source.spm: 100%	<div></div>	842k/842k [00:00<00:00, 30.1MB/s]
target.spm: 100%	<div></div>	813k/813k [00:00<00:00, 30.6MB/s]
vocab.json: 100%	<div></div>	1.72M/1.72M [00:00<00:00, 12.9MB/s]
config.json: 100%	<div></div>	1.38k/1.38k [00:00<00:00, 55.1kB/s]
pytorch_model.bin: 100%	<div></div>	307M/307M [00:07<00:00, 54.6MB/s]
generation_config.json: 100%	<div></div>	293/293 [00:00<00:00, 12.9kB/s]
tokenizer_config.json: 100%	<div></div>	42.0/42.0 [00:00<00:00, 1.89kB/s]

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
Running on public URL: <https://100ca0e9a28ceac748.gradio.live>

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Terminal to deploy to Spaces (<https://huggingface.co/spaces>)

Upload an audio file and get the transcription in English and translation in the selected language.

Explanation (Cell 13):

- ❖ **Import Libraries:** Imports librosa, numpy, and gradio.
- ❖ **Setup transcriber:** Initializes a speech recognition pipeline (transcriber) using a large model "openai/whisper-large" for better performance in recognizing speech.
- ❖ **Setup translation_pipelines:** Creates a dictionary (translation_pipelines) that maps different languages to their respective translation pipelines:
 - ✓ "Dutch", "Spanish", "Korean", and "Russian" are some of the languages configured for the demo purpose to use specific translation models from the transformers library.
- ❖ **Define transcribe_and_translate Function:**
 - ✓ Takes an audio file (audio_file) and a target language (language) as inputs.
 - ✓ Loads the audio file using librosa.load and extracts the sampling rate (sr) and raw audio data (y).
 - ✓ Normalizes the raw audio data to float32 format and scales it.
 - ✓ Performs transcription using transcriber by passing the raw audio data, its sampling rate (sr), and specifying "task": "transcribe".

- ✓ Retrieves the transcription (transcription) from the transcriber output.
- ✓ Translates the transcription into the selected language using the corresponding translation pipeline (translation_pipelines[language]).
- ✓ Returns both the transcription and the translation.

❖ Setup demo Interface:

Uses gr.Interface to create an interface (demo) for the transcribe_and_translate function:

- ✓ Defines transcribe_and_translate as the processing function.
- ✓ Specifies two inputs: gr.Audio(type="filepath") for uploading audio files and gr.Dropdown for selecting the target translation language.
- ✓ Outputs two text boxes (gr.Textbox): one for the transcription in the original language and another for the translation in the selected language.
- ✓ Provides a title ("Multilingual Audio Transcription and Translation") and description for the interface.

- ❖ **Launch Interface:** demo.launch() starts the interface, enabling users to upload an audio file and select a target language for transcription and translation.

#Cell 14

```
import librosa
import numpy as np

transcriber = pipeline("automatic-speech-recognition", model="openai/whisper-large")

translation_pipelines = {
    "Dutch": pipeline("translation", model="Helsinki-NLP/opus-mt-en-nl"),
    "Spanish": pipeline("translation", model="Helsinki-NLP/opus-mt-en-es"),
    "Korean": pipeline("translation", model="Helsinki-NLP/opus-mt-ko-en"),
    "Russian": pipeline("translation", model="Helsinki-NLP/opus-mt-en-ru"),
    "English": pipeline("translation", model="Helsinki-NLP/opus-mt-mul-en")
}

def transcribe_and_translate(audio_file, language):
    try:
        y, sr = librosa.load(audio_file, sr=None)

        y = y.astype(np.float32)
        y /= np.max(np.abs(y))

        transcription = transcriber({"sampling_rate": sr, "raw": y})["text"]
        print(f"Transcription: {transcription}")

        english_translation =
translation_pipelines["English"](transcription)[0]['translation_text']
        print(f"English Translation: {english_translation}")
```



```

        selected_language_translation =
translation_pipelines[language](english_translation)[0]['translation_text']
        print(f"{language} Translation: {selected_language_translation}")

    return transcription, english_translation, selected_language_translation
except Exception as e:
    print(f"Error: {e}")
    return "Error during transcription", "Error during translation to English",
"Error during translation to selected language"

demo = gr.Interface(
    fn=transcribe_and_translate,
    inputs=[
        gr.Audio(type="filepath"),
        gr.Dropdown(label="Select Language", choices=["Dutch", "Spanish", "Korean",
"Russian"], value="Russian")
    ],
    outputs=[
        gr.Textbox(label="Transcription in Original Language"),
        gr.Textbox(label="Translation in English"),
        gr.Textbox(label="Translation in Selected Language")
    ],
    title="Multilingual Audio Transcription and Translation",
    description="Upload an audio file and get the transcription."
)

demo.launch()

```

Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.

config.json: 100%	1.40k/1.40k [00:00<00:00, 84.3kB/s]
pytorch_model.bin: 100%	310M/310M [00:06<00:00, 20.8MB/s]
generation_config.json: 100%	293/293 [00:00<00:00, 19.5kB/s]
tokenizer_config.json: 100%	44.0/44.0 [00:00<00:00, 2.69kB/s]
source.spm: 100%	707k/707k [00:00<00:00, 2.94MB/s]
target.spm: 100%	791k/791k [00:00<00:00, 12.8MB/s]
vocab.json: 100%	1.42M/1.42M [00:00<00:00, 4.43MB/s]

Setting queue=True in a Colab notebook requires sharing enabled. Setting `share=True` (you can turn this off by setting `share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
Running on public URL: <https://25c223fcd226833ab4.gradio.live>

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Terminal to deploy to Spaces (<https://huggingface.co/spaces>)

Explanation (Cell 14):

- ❖ Imports librosa for audio loading and processing.
- ❖ Imports numpy for numerical operations.
- ❖ Sets up a speech recognition pipeline (transcriber) using the pipeline function from transformers.
- ❖ Specifies "openai/whisper-large" as the model, which is optimized for accurate automatic speech recognition.
- ❖ Initializes a dictionary translation_pipelines that maps languages to their respective translation pipelines.
- ❖ Each pipeline is created using the pipeline function from transformers, specifying different translation models for each language.
- ❖ **Input:** Takes audio_file (path to the audio file) and language (target language for translation) as parameters.
- ❖ **Steps:**
 - i. **Audio Processing:**
 - ✓ Loads the audio file (audio_file) using librosa.load and extracts the raw audio (y) and its sampling rate (sr).
 - ✓ Converts the raw audio to float32 and normalizes it to ensure consistent amplitude levels.
 - ii. **Transcription:**
 - ✓ Uses the transcriber pipeline to transcribe the normalized audio (y) into text (transcription).
 - ✓ Prints the transcription to provide feedback during processing.
 - iii. **Translation:**
 - ✓ Translates the transcription into English using translation_pipelines["English"].
 - ✓ Prints the English translation for feedback.
 - ✓ Translates the English translation into the specified language using translation_pipelines[language].
 - ✓ Prints the translation in the selected language for feedback.
 - iv. **Error Handling:**
 - ✓ Catches and handles any exceptions that may occur during audio processing, transcription, or translation. Prints the specific error message for each stage.
- ❖ **Output:** Returns three outputs: transcription, english_translation, and selected_language_translation.

```
# Cell 15

import numpy as np
import librosa
from difflib import SequenceMatcher

RAG_DOCUMENT = {
    "What is the capital of France?": "The capital of France is Paris.",
```

```

    "Who wrote 'To Kill a Mockingbird'?": "'To Kill a Mockingbird' was written by Harper Lee.",
    "What is the largest planet in our solar system?": "The largest planet in our solar system is Jupiter.",
    "What is the boiling point of water in Celsius?": "The boiling point of water is 100 degrees Celsius.",
    "Who painted the Mona Lisa?": "The Mona Lisa was painted by Leonardo da Vinci.",
    "What year did the Titanic sink?": "The Titanic sank in 1912.",
    "Who discovered penicillin?": "Penicillin was discovered by Alexander Fleming.",
    "What is the tallest mountain in the world?": "Mount Everest is the tallest mountain in the world.",
    "Who invented the telephone?": "The telephone was invented by Alexander Graham Bell.",
    "What is the currency of Japan?": "The currency of Japan is the Japanese Yen.",
    "What is the speed of light in a vacuum?": "The speed of light in a vacuum is approximately 299,792 kilometers per second.",
}

transcriber = pipeline("automatic-speech-recognition", model="openai/whisper-base.en")

def find_best_match(query, documents):
    best_match = None
    highest_ratio = 0.0
    for question in documents:
        ratio = SequenceMatcher(None, query, question).ratio()
        if ratio > highest_ratio:
            highest_ratio = ratio
            best_match = question
    return best_match

def transcribe_and_query(audio_file):
    y, sr = librosa.load(audio_file, sr=None)
    y = y.astype(np.float32)
    y /= np.max(np.abs(y))

    transcription = transcriber({"sampling_rate": sr, "raw": y})["text"]

    best_match = find_best_match(transcription, RAG_DOCUMENT)

    if best_match and SequenceMatcher(None, transcription, best_match).ratio() > 0.7:
        answer = RAG_DOCUMENT[best_match]
    else:
        answer = "Sorry, I couldn't find the answer."

    return transcription, answer

# Gradio interface
demo = gr.Interface(
    fn=transcribe_and_query,
    inputs=gr.Audio(type="filepath"),

```

```

outputs=[gr.Textbox(label="Transcription"), gr.Textbox(label="Answer")],
title="RAG Document Query with Speech",
description="Upload an audio file with a question and get the answer from the
RAG document."
)

demo.launch()

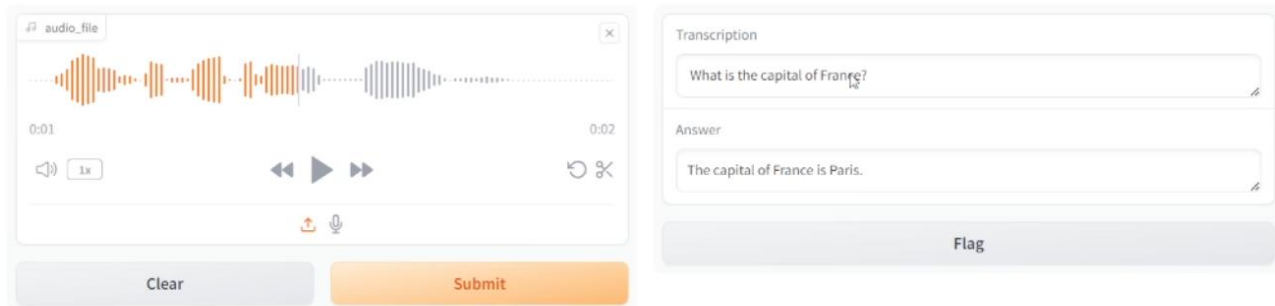
```

Setting queue=True in a Colab notebook requires sharing enabled. Setting `share=True` (you can turn this off by setting `share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()

Running on public URL: <https://f3ca9e7c9b4d7f5f96.gradio.live>

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Terminal to deploy to Spaces (<https://huggingface.co/spaces>)



Use via API • Built with Gradio

Explanation (Cell 15):

- ✓ Imports necessary libraries:
 - ✓ numpy for numerical operations.
 - ✓ librosa for audio processing.
 - ✓ SequenceMatcher from difflib for comparing string similarity.
- ✓ Defines RAG_DOCUMENT, a dictionary mapping questions to their corresponding answers. This serves as the knowledge base for the question answering system.

Speech Recognition Setup:

- ✓ Initializes the transcriber pipeline using `pipeline("automatic-speech-recognition", model="openai/whisper-base.en")`. This pipeline is configured to perform automatic speech recognition using the openai/whisper-base.en model.

Utility Functions:

- ✓ **find_best_match Function:**
 - ✓ Takes a query (transcription of audio) and a documents dictionary (RAG_DOCUMENT).
 - ✓ Iterates through each question in documents, computes the similarity ratio between query and each question using SequenceMatcher.
 - ✓ Returns the question from documents with the highest similarity ratio to query.

✓ **Main Processing Function (transcribe_and_query):**

Input: audio_file (path to the audio file to be transcribed and queried).

Steps:

- ✓ Loads the audio file (audio_file) using librosa.load and extracts the sampling rate (sr) and raw audio data (y).
- ✓ Normalizes y to float32 format and scales it.
- ✓ Transcribes the audio using transcriber, passing sr and y as parameters. Retrieves the transcription (transcription) from the output.
- ✓ Finds the best matching question (best_match) from RAG_DOCUMENT based on transcription.
- ✓ Checks if best_match exists and if the similarity ratio between transcription and best_match is above 0.7.
- ✓ If conditions are met, retrieves the corresponding answer from RAG_DOCUMENT. Otherwise, returns a default message indicating the answer couldn't be found.

✓ **Gradio Interface Setup (demo):**

Uses gr.Interface to define a graphical interface (demo) for the transcribe_and_query function:

- ✓ Specifies transcribe_and_query as the processing function.
- ✓ Sets gr.Audio(type="filepath") as the input type, allowing users to upload an audio file.
- ✓ Defines two outputs: gr.Textbox(label="Transcription") for displaying the transcription and gr.Textbox(label="Answer") for displaying the answer.
- ✓ Provides a title ("RAG Document Query with Speech") and description for the interface.

✓ **Launching the Interface:**

demo.launch() starts the Gradio interface, enabling users to upload an audio file containing a question. Upon submission, the interface transcribes the audio, matches it to the closest question in RAG_DOCUMENT, and displays both the transcription and the corresponding answer.

Demo Link : <https://youtu.be/fptOkUeIIKY>