**Hindi Vidya Prachar Samiti's**

**Ramniranjan Jhunjhunwala College of Arts, Science and Commerce**

**(EMPOWERED AUTONOMOUS)**

**Project Report on**

**Cross-Lingual Audio Similarity via Embeddings**

**Submitted in Partial fulfilment of the Requirements for the award of the Degree of Master in Science in Data Science & Artificial Intelligence**

**By**

**Miss. Dipti Suchiwarta Waghmare**

**Miss. Zainab Hamid Husain Qazi**

**Seat no.:10409 & 10402**

**Under the guidance of,**

**Prof. Mujtaba Shaikh**

**Designation, Dept. of DSAI**

**Department of Data Science & Artificial Intelligence**

**(Affiliated to University of Mumbai)**

**Hindi Vidya Prachar Samiti's Ramniranjan Jhunjhunwala College of Arts,**

**Commerce and Science (EMPOWERED AUTONOMOUS)**

**Ghatkopar (West), Mumbai-86**

**2024-25**

# RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ART’S, SCIENCE & COMMERCE (AUTONOMOUS), GHATKOPAR (W)



***(Affiliated to University of Mumbai)***

# Certificate



*This is to certify that the Project entitled* **Cross-Lingual Audio Similarity via Embeddings** *is bonafide work of* **Miss Dipti Suchiwarta Waghmare** *bearing Seat No* **10409** *submitted in partial fulfilment of the requirements for the award of Degree* ***Master of Science*** *in* ***Data Science & Artificial Intelligence.***

**Signature of Internal Guide Signature of Co-ordinator**

**College Seal Signature of Examiner**

**Date:**

# Table of Contents

|  |  |  |
| --- | --- | --- |
| **Sr No.** | **Title** | Page No. |
| **1.** | Acknowledgement | 1 |
| **2.** | Abstract | 2 |
| **3.** | Introduction | 3 |
| **4.** | Literature Review | 5 |
| **5.** | Methodology | 7 |
| **6.** | Block Diagram | 9 |
| **7.** | Setting Up Environment | 10 |
| **8.** | Datasets | 13 |
| **9** | Code | 14 |
| **10.** | Deployment | 24 |
| **11.** | Snapshot | 25 |
| **12.** | Limitations | 28 |
| **13.** | Future Scope | 29 |
| **14.** | Conclusion | 30 |
| **15.** | References | 31 |

## Acknowledgement

Before we get into thick of things, we would like to add a few heartfelt words for the people who were part of **Cross-Lingual Audio Similarity via Embeddings** project in numerous ways, people who gave unending support right from the stage the project idea was conceived.

A project report is such a comprehensive coverage; it would not have been materialized without the help of many. The four things that go on to make a successful endeavour are dedication, hard work, patience and correct guidance. Able and timely guidance not only helps in making an effort fruitful but also transforms the whole process of learning and implementing into an enjoyable experience.

In particular, I would like to thank our Director **Dr. (Mrs.) Usha Mukundan**, R.J. College. I would like to give a very special honor and respect to our teacher, **Prof. Mujtaba Shaikh** who took keen interest in checking the minute details of the project work and guided us throughout the same. A sincere quote of thanks to the non-teaching staff for providing us software their time. I appreciate outstanding co-operation by them, especially for the long Lab timings that we could receive.

## Abstract

With the growing demand for multilingual applications and global communication, there is an increasing need for voice technologies that can compare and match voices across different languages. This project addresses that challenge by developing a system capable of evaluating the similarity between two voice samples, even when the speakers are using different languages. Unlike traditional speaker verification systems, our approach can determine whether two voices sound alike, regardless of whether they belong to the same person.

We focus on English and Hindi speech data and leverage deep learning-based audio embeddings using the pre-trained PyAnnote model. This model captures high-level, language-agnostic features from voice recordings. To quantify similarity, we compute cosine similarity between the extracted embeddings. A higher similarity score indicates that the two voices share similar vocal characteristics.

The system is integrated into a user-friendly Streamlit web application that allows users to upload two audio files and receive an instant similarity score. Audio pre-processing steps include resampling, mono conversion, and duration normalization (via trimming or padding). The audio files are then converted into embeddings, normalized, and compared.

We use the Common Voice dataset multilingual open-source speech corpus for training and testing. The application supports various real-world use cases, such as cross-lingual voice verification, voice-based user interfaces, and speaker matching for media localization or dubbing.

While the system shows promising results, especially for English-Hindi comparisons, limitations such as sensitivity to noise and dataset diversity remain challenges. Nonetheless, this project demonstrates the potential of embedding-based voice comparison for cross-lingual and multilingual applications and lays the groundwork for future improvements in this space.

## Introduction

In recent years, voice-based technologies have emerged as a central component of human-computer interaction, playing a crucial role in applications ranging from smart assistants (such as Amazon Alexa, Apple Siri, and Google Assistant) to security systems, call center automation, and accessibility tools. The increasing reliance on voice interfaces has driven significant advances in speaker recognition—enabling systems to accurately identify or verify individuals based on their speech. However, most of these systems have been optimized for monolingual scenarios, where both the training and testing data are derived from the same language. This poses a limitation in global, multilingual settings where people often switch between languages or communicate in multiple languages throughout their day.

One particularly intriguing challenge in this space is cross-lingual voice similarity detection—the ability to compare two voice recordings, possibly in different languages, and evaluate how similar the speakers sound. This goes beyond simply identifying whether the speakers are the same individual. It includes evaluating how closely two different voices resemble one another, based on their acoustic and paralinguistic features such as tone, pitch, vocal texture, intonation, and rhythm, irrespective of the linguistic content of the speech. Such a system can, for example, compare a speaker speaking English to another speaker speaking Hindi and estimate how similar their voices are—even if the languages, words, and phonetic structures are completely different.

This capability has numerous real-world applications. In multilingual societies like India, where code-switching is common and citizens frequently speak multiple languages, systems that can compare and retrieve voices across languages can play a critical role in biometric authentication, voice-based search, digital assistants, and even in areas like casting for voice dubbing in media production. Moreover, in industries such as law enforcement and forensic analysis, having the ability to assess voice similarity without language barriers can be a powerful tool for speaker profiling and identity verification.

However, building such a system is non-trivial. Traditional speaker recognition models, which often rely on handcrafted features or language-dependent phonetic models, struggle when confronted with language variation. Differences in pronunciation, prosody, and phonemic inventory across languages introduce variability that can confound these systems. Furthermore, conventional voice comparison methods often assume that two recordings are in the same language, making them unsuitable for cross-lingual scenarios.

To overcome these challenges, this project leverages deep learning-based audio embedding models, which have demonstrated the ability to extract language-independent speaker representations. Specifically, we use PyAnnote, a pre-trained model known for producing robust speaker embeddings used in speaker diarization and recognition tasks. These embeddings represent voice recordings as fixed-length, high-dimensional vectors that encapsulate a speaker’s unique vocal characteristics, largely invariant to the spoken language. This approach allows us to analyze and compare voices at a deeper, more abstract level, focusing on the inherent vocal traits rather than the linguistic content.

The audio samples for this project are sourced from the Common Voice dataset, an open-source multilingual speech corpus developed by Mozilla. We focus on English and Hindi recordings, pre-processing them through normalization, silence trimming, mono-channel conversion, and fixed-duration clipping or padding to ensure consistency. These processed audio files are then passed through the PyAnnote model to extract embeddings.

To quantify voice similarity, we use cosine similarity, a widely adopted metric in embedding space analysis. Cosine similarity calculates the cosine of the angle between two embedding vectors, yielding a score between -1 and 1, where higher values indicate greater similarity. This score serves as an intuitive and effective measure of how acoustically alike two voices sound.

To make our system accessible and user-friendly, we have developed a web-based application using Streamlit, a powerful and lightweight framework for building interactive data science apps. The application allows users to upload two audio files in real time and instantly receive a similarity score along with visual feedback, making it an ideal tool for demonstrations, experiments, and even deployment in real-world systems.

The implications of this project extend beyond the immediate task of voice comparison. By enabling accurate, language-agnostic voice similarity estimation, we contribute to a growing body of work in multilingual and cross-lingual speech processing, paving the way for future research and applications in multilingual speaker verification, cross-cultural human-computer interaction, voice-driven search engines, and inclusive voice interfaces that work across linguistic boundaries.

While this project presents promising results, we also acknowledge current limitations, such as potential sensitivity to background noise, varying recording conditions, and the inherent bias introduced by the dataset's speaker diversity. These aspects will be important areas for future improvement, including incorporating noise-robust features, domain adaptation techniques, and larger, more diverse multilingual datasets.

In conclusion, this project explores a novel yet essential area of voice technology by bridging the gap between speaker recognition and voice similarity across languages. By combining deep learning, high-quality multilingual datasets, and user-centric design, we aim to build a system that is not only technically robust but also practically valuable in a world that is increasingly connected through diverse languages and voices.

## Literature Review

1. **Audio-Based Near-Duplicate Video Retrieval with Audio Similarity Learning**This study applied similarity learning to identify near-duplicate videos based on audio content. The architecture demonstrated robustness to variations like speed changes in audio and highlighted the effectiveness of transfer learning. Although the dataset was visually annotated, the approach shows potential for tasks such as audio fingerprinting and cover song detection—indicating that audio embeddings can capture perceptual similarity beyond identical content.

<https://arxiv.org/abs/2010.08737>

1. **Content-Based Representations of Audio Using Siamese Neural Networks**This work introduced a Siamese Neural Network architecture to learn semantic audio representations. These embeddings were shown to be effective for content-based audio retrieval, capturing similarities between clips containing the same or similar audio events. Both Euclidean and cosine similarity performed well, underscoring the robustness of the learned representations.

<https://arxiv.org/abs/1710.10974>

1. **EACeleb: An East Asian Language Speaking Celebrity Dataset**

The EACeleb dataset provides a rich resource for training speaker recognition systems tailored to East Asian languages. It demonstrates how diarization and pre-processing can clean and structure large-scale audio from platforms like YouTube. The dataset supports fine-tuning of existing models, enhancing performance for specific language groups—an approach that could be adapted for multilingual or regional speaker modelling.

[**https://arxiv.org/abs/2203.05333**](https://arxiv.org/abs/2203.05333)

1. **Audio Similarity Detection**

Audio similarity detection identifies and measures how alike two or more audio signals are. It is used in applications like speaker recognition, content retrieval, and voice casting. The main challenge lies in selecting features that reflect specific similarity types—semantic, perceptual, or identity-based. Deep learning models help by learning embeddings that capture complex audio traits. Metrics like cosine similarity and contrastive loss are commonly used. Research focuses on matching features and distance metrics to the specific audio similarity task at hand.

<http://www.apsipa2024.org/files/papers/478.pdf>

1. **PyAnnote: Speaker Embedding for Speaker Diarization (HuggingFace)**

PyAnnote provides state-of-the-art speaker embeddings, pre-trained on large and diverse speech corpora. Originally developed for speaker diarization, its embeddings capture speaker-specific features that generalize well across different languages and environments. This makes PyAnnote a suitable backbone for voice similarity systems in multilingual settings.

## Methodology

This chapter outlines the systematic approach followed in the development of a cross-lingual voice similarity system. This project follows a structured pipeline combining signal processing, deep learning, and web interface development. The methodology comprises four primary stages: audio pre-processing, embedding extraction, similarity computation, and user interaction via a web interface. The workflow is designed to be both technically robust and user-friendly.

### **Historical Background**

Traditional speaker recognition systems primarily relied on statistical models such as Gaussian Mixture Models (GMMs) and i-vectors, which required manual feature extraction using audio descriptors like Mel-Frequency Cepstral Coefficients (MFCCs). These techniques, while effective in controlled settings, demonstrated limited generalization across varying acoustic environments and languages.

With the advent of deep learning, representation learning techniques gained popularity. Embedding-based models, including x-vectors and the PyAnnote framework, provided a robust solution by learning speaker-specific characteristics directly from the data. PyAnnote’s pre-trained models, trained on large and diverse corpora, produce high-dimensional vector embeddings that capture speaker identity in a language-agnostic manner.

Cosine similarity, a long-established metric in speaker verification, allows for a straightforward comparison of these embeddings by measuring the angular distance between vectors in high-dimensional space.

### **Audio Pre-processing**

To ensure consistency and compatibility with the PyAnnote embedding model, all audio files undergo a standardized pre-processing pipeline:

* **Mono Conversion:** All audio files are converted to mono-channel to maintain uniformity.
* **Resampling:** Audio is resampled to 16kHz, which aligns with the training specifications of the embedding model.
* **Duration Standardization:** Audio clips are trimmed or zero-padded to a fixed length of 5 seconds. This step ensures that all inputs are of equal size for embedding extraction.

### **Embedding Extraction**

We use the pyannote/embedding model from the PyAnnote library, which generates fixed-length embeddings from audio inputs. The steps involved are:

* Input the pre-processed audio into the PyAnnote model.
* Extract a 1-dimensional embedding vector (typically 512-dimensional) representing the speaker’s vocal identity.
* These embeddings serve as compact representations that encode vocal traits such as pitch, tone, timbre, and speaking style, independent of the spoken language.

### **Similarity Computation**

To compare the two voice embeddings, we use **cosine similarity**, defined as:

Where:

* A and B are the embedding vectors of the two audio samples.
* The result lies between **-1** and **1**, where:  
  + **1** indicates identical vectors (maximum similarity),
  + **0** indicates orthogonality (no similarity),
  + **-1** indicates opposite vectors (very rare in speaker comparison).

This score is used to determine how similar the two voices sound, regardless of language or speaker identity.

### **Web Interface Implementation**

The entire pipeline is encapsulated within a user-friendly interface built using **Streamlit**, a Python-based framework for creating interactive web applications. The interface provides the following functionalities:

* **Audio File Upload:** Users can upload two audio files through a graphical interface.
* **Real-time Similarity Computation:** Once the audio files are processed, the cosine similarity score is computed and displayed.

## Block Diagram

**Cosine Similarity Computation**

**Similarity Score UI**

**Audio File A**

**Audio File B**

**Pre-processing**

(Mono,16kHz Trim/Pad to 5 seconds)

**Embedding A**

**Embedding B**

## Setting up the Environment

A suitable development environment is essential for implementing and deploying the voice similarity detection system effectively. This chapter outlines the required tools, libraries, and hardware specifications necessary for setting up the project.

### **Python Environment**

* **Python Version:** 3.10 or higher  
   Python 3.10 is recommended due to its compatibility with the latest versions of essential machine learning and audio processing libraries.

### **Required Libraries and Frameworks**

The following Python libraries are used throughout the project:

* **librosa** A Python package for music and audio analysis. It is used for loading, resampling, and pre-processing audio files.
* **torch (PyTorch)** A deep learning framework that powers the pyannote.audio model. It enables efficient tensor computation and model inference.
* **pyannote.audio** A pre-trained audio embedding model specifically designed for speaker diarization and recognition. It extracts speaker-related features from audio.
* **scikit-learn** A machine learning toolkit used for similarity computations, including cosine similarity, and for data normalization or evaluation utilities.
* **streamlit** A lightweight Python library for building interactive web apps. In this project, it’s used to create the user interface for uploading audio files and displaying similarity results.

**API Keys and Accounts:** Hugging Face API Key

Scenify uses Hugging Face APIs for model interaction, specifically for

image generation. You will need to sign up on Hugging Face and get an

API key to use the models.

Go to your Hugging Face Account → Create a new account or sign

in → Navigate to Settings → Access Tokens → Generate a new

token.

Include the token in your model

### **Installation Instructions**

All dependencies are listed in the requirements.txt file. The project environment can be set up using the following command:

*pip install -r requirements.txt*

This command automatically installs all necessary packages and ensures version consistency across different setups.

### **Hardware Requirements**

While the system can run on a **standard CPU**, a **GPU** is recommended for faster embedding extraction and inference, especially when handling longer or multiple audio files.

|  |  |
| --- | --- |
| Component | Recommended |
| CPU | Intel i5/i7 or AMD Ryzen equivalent |
| GPU | NVIDIA GPU with CUDA support (e.g., GTX 1660 or higher) |
| RAM | 8 GB minimum (16 GB recommended) |
| Storage | SSD preferred for faster I/O operations |

### **Project Structure**

The project directory is organized to support modular development, easy debugging, and streamlined deployment. Below is a breakdown of the folder structure and the role of each key component:

Research Project/│

* .ipynb\_checkpoints/ #Jupyter Notebook checkpoints
* \_\_pycache\_\_/ # Python bytecode cache
* commonvoice\_en/ # Raw English Common Voice dataset
* commonvoice\_hi/ # Raw Hindi Common Voice dataset
* env/ # Python virtual environment
* processed\_en/ # Pre-processed English audio files
* processed\_hi/ # Pre-processed Hindi audio files
* app\_streamlit.py # Streamlit web application script for deployment
* Audio\_Pre-processing.ipynb # Jupyter Notebook for audio data cleaning and transformation
* audio\_utils.py # Core utilities for audio embedding and similarity scoring
* cv-corpus-21.0-\*.tar.gz # Downloaded Common Voice corpora (English & Hindi)
* Model\_cv.ipynb # Notebook for model development, training, and evaluation
* requirements.txt # Python package dependencies

## Dataset

### **Dataset Overview**

* **Name:** Common Voice
* **Source:** [Mozilla Common Voice](https://commonvoice.mozilla.org)
* **Languages Used:** English and Hindi
* **File Format:** .wav (Waveform Audio File Format)

### **Description**

Common Voice is an open-source, multilingual speech dataset developed by Mozilla with the aim of supporting research in speech recognition and speaker identification. This dataset is unique in that it is contributed by volunteers globally, helping to create a diverse and inclusive corpus of audio recordings. Common Voice is particularly valuable for speech processing applications as it contains audio clips from a wide range of speakers, varying in accent, gender, age, and dialect. Each clip is accompanied by metadata such as the speaker ID, the language spoken, and the corresponding text transcription, making it an invaluable resource for training machine learning models in tasks related to speech recognition, language identification, and speaker verification.

Common Voice includes recordings in multiple languages, with English and Hindi being two of the primary languages. The dataset’s open nature allows researchers, developers, and enthusiasts to freely access and utilize it for their projects, furthering advancements in the field of speech technology.

### **Usage in This Project**

For this system, we:

* Selected speaker-specific audio clips from both English and Hindi subsets of the dataset.
* Filtered and organized the clips to ensure diversity in speakers and languages.
* Pre-processed the audio files by:  
  + Converting stereo to mono
  + Resampling to 16kHz
  + Trimming or padding each clip to 5 seconds

These processed audio clips were then passed through the PyAnnote embedding model to extract speaker embeddings for similarity comparison

## Code

#### **Code Component Responsibilities**

* **Audio\_Pre-processing.ipynb**:  
  Contains scripts for:
  + Cleaning audio files.
  + Resampling and normalizing waveforms.
  + Saving them into the processed\_\* directories.

**import tarfile**

**import os**

**def extract\_dataset(tar\_path, extract\_to):**

**with tarfile.open(tar\_path, 'r:gz') as tar:**

**members = tar.getmembers()**

**lang\_code = os.path.basename(tar\_path).split('-')[-1].replace('.tar.gz', '')**

**root\_prefix = None**

**for m in members:**

**if f"/{lang\_code}/" in m.name:**

**root\_prefix = m.name.split(f"{lang\_code}/")[0] + f"{lang\_code}/"**

**break**

**if root\_prefix is None:**

**print(f"Couldn't determine folder structure for: {tar\_path}")**

**return**

**os.makedirs(extract\_to, exist\_ok=True)**

**for member in members:**

**if member.name.startswith(root\_prefix):**

**member.name = os.path.relpath(member.name, root\_prefix)**

**if member.name == '.':**

**continue**

**tar.extract(member, path=extract\_to)**

**print(f"Extracted {lang\_code.upper()} to: {extract\_to} (flattened)")**

**extract\_dataset(**

**"cv-corpus-21.0-delta-2025-03-14-en.tar.gz",**

**r"C:\Users\WAGHMARE\Desktop\Research Project\commonvoice\_en"**

**)**

**extract\_dataset(**

**"cv-corpus-21.0-2025-03-14-hi.tar.gz",**

**r"C:\Users\WAGHMARE\Desktop\Research Project\commonvoice\_hi"**

**)**

**pip install pydub tqdm**

**import os**

**clips\_folder = r"C:\Users\WAGHMARE\Desktop\Research Project\commonvoice\_hi\clips"**

**validated\_tsv = r"C:\Users\WAGHMARE\Desktop\Research Project\commonvoice\_hi\validated.tsv"**

**existing\_files = set(os.listdir(clips\_folder))**

**with open(validated\_tsv, 'r', encoding='utf-8') as f:**

**lines = f.readlines()**

**header = lines[0]**

**valid\_lines = [header]**

**for line in lines[1:]:**

**parts = line.strip().split('\t')**

**if len(parts) > 1:**

**filename = parts[1]**

**if filename in existing\_files:**

**valid\_lines.append(line)**

**output\_filtered\_tsv = os.path.join(**

**r"C:\Users\WAGHMARE\Desktop\Research Project\commonvoice\_hi", "validated\_filtered.tsv"**

**)**

**with open(output\_filtered\_tsv, 'w', encoding='utf-8') as f:**

**f.writelines(valid\_lines)**

**print(f"Filtered validated.tsv to {len(valid\_lines)-1} entries with existing files.")**

**import os**

**clips\_folder = r"C:\Users\WAGHMARE\Desktop\Research Project\commonvoice\_en\clips"**

**validated\_tsv = r"C:\Users\WAGHMARE\Desktop\Research Project\commonvoice\_en\validated.tsv"**

**existing\_files = set(os.listdir(clips\_folder))**

**with open(validated\_tsv, 'r', encoding='utf-8') as f:**

**lines = f.readlines()**

**header = lines[0]**

**valid\_lines = [header]**

**for line in lines[1:]:**

**parts = line.strip().split('\t')**

**if len(parts) > 1:**

**filename = parts[1]**

**if filename in existing\_files:**

**valid\_lines.append(line)**

**output\_filtered\_tsv = os.path.join(**

**r"C:\Users\WAGHMARE\Desktop\Research Project\commonvoice\_en", "validated\_filtered.tsv"**

**)**

**with open(output\_filtered\_tsv, 'w', encoding='utf-8') as f:**

**f.writelines(valid\_lines)**

**print(f"Filtered validated.tsv to {len(valid\_lines)-1} entries with existing files.")**

**import os**

**from pydub import AudioSegment**

**from pydub.utils import which**

**ffmpeg\_path = r"C:\Users\WAGHMARE\Downloads\ffmpeg-7.1.1-essentials\_build\bin\ffmpeg.exe"**

**AudioSegment.converter = ffmpeg\_path**

**os.environ["PATH"] += os.pathsep + os.path.dirname(ffmpeg\_path)**

**print("FFmpeg manually set to:", AudioSegment.converter)**

**print("FFmpeg found by which():", which("ffmpeg"))**

**import os**

**import pandas as pd**

**import shutil**

**from pydub import AudioSegment**

**from tqdm import tqdm**

**def prepare\_data(language\_folder, output\_folder, max\_samples\_per\_speaker=10):**

**validated\_file = os.path.join(language\_folder, "validated\_filtered.tsv")**

**if not os.path.exists(validated\_file):**

**print(f"Error: {validated\_file} does not exist!")**

**return**

**df = pd.read\_csv(validated\_file, sep="\t")**

**if 'client\_id' not in df.columns or 'path' not in df.columns:**

**print("Error: validated\_filtered.tsv missing required columns.")**

**return**

**# Drop rows with missing 'path' or 'client\_id'**

**df = df[df['path'].notnull() & df['client\_id'].notnull()]**

**# Group by speaker and limit samples**

**grouped = df.groupby("client\_id").head(max\_samples\_per\_speaker)**

**print(f"Total unique speakers in {language\_folder}: {df['client\_id'].nunique()}")**

**print(f"Preparing data for {grouped['client\_id'].nunique()} speakers")**

**clips\_folder = os.path.join(language\_folder, "clips")**

**if not os.path.exists(clips\_folder):**

**print(f"Error: Clips folder not found at {clips\_folder}")**

**return**

**os.makedirs(output\_folder, exist\_ok=True)**

**for \_, row in tqdm(grouped.iterrows(), total=len(grouped), desc="Converting MP3 to WAV"):**

**client\_id = row['client\_id']**

**filename = row['path']**

**mp3\_path = os.path.join(clips\_folder, filename)**

**if not os.path.exists(mp3\_path):**

**print(f"File missing: {mp3\_path}")**

**continue**

**speaker\_folder = os.path.join(output\_folder, client\_id)**

**os.makedirs(speaker\_folder, exist\_ok=True)**

**wav\_filename = filename.replace(".mp3", ".wav")**

**wav\_path = os.path.join(speaker\_folder, wav\_filename)**

**try:**

**audio = AudioSegment.from\_mp3(mp3\_path)**

**audio.export(wav\_path, format="wav")**

**except Exception as e:**

**print(f"Error converting {mp3\_path}: {e}")**

**print("Data preparation complete.")**

**prepare\_data(**

**language\_folder=r"C:\Users\WAGHMARE\Desktop\Research Project\commonvoice\_hi",**

**output\_folder=r"C:\Users\WAGHMARE\Desktop\Research Project\processed\_hi",**

**max\_samples\_per\_speaker=10**

**)**

**prepare\_data(**

**language\_folder=r"C:\Users\WAGHMARE\Desktop\Research Project\commonvoice\_en",**

**output\_folder=r"C:\Users\WAGHMARE\Desktop\Research Project\processed\_en",**

**max\_samples\_per\_speaker=10**

**)**

* **Model\_cv.ipynb**:  
  Used primarily for:
  + Loading datasets and analyzing them.
  + Testing embedding functions.
  + Visualizing performance metrics.

**import glob**

**import os**

**import librosa**

**import torch**

**import numpy as np**

**from pyannote.audio import Inference**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.metrics.pairwise import cosine\_similarity**

**from tqdm import tqdm**

**device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")**

**print(f"Using device: {device}")**

**inference = Inference("pyannote/embedding",**

**use\_auth\_token="hf\_UvluopJYmdTpeerHDLohdXdUXwNpCCFAQJ",**

**window="whole",**

**device=device)**

**MAX\_DURATION = 5.0**

**def extract\_embeddings\_from\_dir(directory):**

**wav\_files = glob.glob(os.path.join(directory, "\*\*", "\*.wav"), recursive=True)**

**print(f"\n Found {len(wav\_files)} audio files in {directory}")**

**embeddings = []**

**file\_labels = []**

**for path in tqdm(wav\_files, desc=f"Processing {os.path.basename(directory)}"):**

**try:**

**waveform, sr = librosa.load(path, sr=16000, mono=True)**

**if len(waveform) > MAX\_DURATION \* sr:**

**waveform = waveform[:int(MAX\_DURATION \* sr)]**

**else:**

**pad\_len = int(MAX\_DURATION \* sr) - len(waveform)**

**waveform = np.pad(waveform, (0, pad\_len), 'constant')**

**inputs = {**

**"waveform": torch.tensor(waveform, dtype=torch.float32).unsqueeze(0).to(device),**

**"sample\_rate": sr**

**}**

**with torch.no\_grad():**

**embedding = inference(inputs)**

**embeddings.append(embedding.flatten())**

**file\_labels.append(path)**

**except Exception as e:**

**print(f"Error processing {path}: {e}")**

**return embeddings, file\_labels**

**ENGLISH\_DIR = r"C:\Users\WAGHMARE\Desktop\Research Project\processed\_en"**

**HINDI\_DIR = r"C:\Users\WAGHMARE\Desktop\Research Project\processed\_hi"**

**embeddings\_en, labels\_en = extract\_embeddings\_from\_dir(ENGLISH\_DIR)**

**embeddings\_hi, labels\_hi = extract\_embeddings\_from\_dir(HINDI\_DIR)**

**def normalize\_embedding(embedding):**

**return (embedding - np.mean(embedding)) / np.std(embedding)**

**def compare\_audio\_files(file1, file2, inference\_model, device, duration=5.0):**

**def process\_audio(path):**

**waveform, sr = librosa.load(path, sr=16000, mono=True)**

**if len(waveform) > duration \* sr:**

**waveform = waveform[:int(duration \* sr)]**

**else:**

**pad\_length = int(duration \* sr) - len(waveform)**

**waveform = np.pad(waveform, (0, pad\_length), 'constant')**

**inputs = {**

**"waveform": torch.tensor(waveform, dtype=torch.float32).unsqueeze(0).to(device),**

**"sample\_rate": sr**

**}**

**with torch.no\_grad():**

**embedding = inference\_model(inputs)**

**if isinstance(embedding, dict):**

**embedding = embedding["embedding"]**

**if isinstance(embedding, torch.Tensor):**

**embedding = embedding.detach().cpu().numpy()**

**embedding = embedding.flatten()**

**return embedding**

**emb1 = process\_audio(file1)**

**emb2 = process\_audio(file2)**

**emb1\_normalized = normalize\_embedding(emb1)**

**emb2\_normalized = normalize\_embedding(emb2)**

**print(f"Embedding 1 (First 10 values): {emb1\_normalized[:10]}")**

**print(f"Embedding 2 (First 10 values): {emb2\_normalized[:10]}")**

**similarity = cosine\_similarity([emb1\_normalized], [emb2\_normalized])[0][0]**

**# Print the similarity score**

**print(f"\n Similarity between:\n'{file1}'\nand\n'{file2}' ➜ {similarity:.4f}")**

**return similarity**

* **audio\_utils.py**:  
  Contains reusable functions to:
  + Generate embeddings from audio files.
  + Calculate cosine similarity scores.
  + Perform other audio signal operations.

import librosa

import torch

import numpy as np

from pyannote.audio import Inference

from sklearn.metrics.pairwise import cosine\_similarity

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

inference = Inference("pyannote/embedding", device=device)

def extract\_embedding(file\_path):

    return inference(file\_path).data.mean(axis=0).reshape(1, -1)

def get\_similarity\_score(embedding1, embedding2):

    return float(cosine\_similarity(embedding1, embedding2)[0][0])

* **app\_streamlit.py**:  
  A lightweight interface built using Streamlit that:
  + Takes two audio files as input.
  + Computes and displays their similarity score using the trained model.

import streamlit as st

from audio\_utils import extract\_embedding, get\_similarity\_score

import os

st.set\_page\_config(page\_title="Audio Similarity Checker", layout="centered")

st.title("🔊 Audio Similarity Checker")

uploaded\_file1 = st.file\_uploader("Upload first audio file", type=["wav", "mp3"])

uploaded\_file2 = st.file\_uploader("Upload second audio file", type=["wav", "mp3"])

if uploaded\_file1 and uploaded\_file2:

    with open("temp1.wav", "wb") as f:

        f.write(uploaded\_file1.read())

    with open("temp2.wav", "wb") as f:

        f.write(uploaded\_file2.read())

    if st.button("🎯 Check Similarity"):

        st.info("✅ Files uploaded. Extracting embeddings...")

        emb1 = extract\_embedding("temp1.wav")

        emb2 = extract\_embedding("temp2.wav")

        score = get\_similarity\_score(emb1, emb2)

        st.subheader(f"🧠 Similarity Score: `{score:.3f}`")

        if score > 0.7:

            st.success("🎧 The audios are similar!")

        else:

            st.warning("🛑 The audios are NOT similar.")

        os.remove("temp1.wav")

        os.remove("temp2.wav")

## Deployment

The application is deployed using **Streamlit**, a lightweight and efficient open-source framework designed for building interactive web applications in Python, particularly suited for data science and machine learning projects.

**Deployment Process**

The app is encapsulated in a single script: app\_streamlit.py. This script integrates all key components, including the user interface, audio preprocessing, speaker embedding extraction, and similarity computation.

To run the deployed application, the following command is used:

*streamlit run app\_streamlit.py*

This command launches a local web server, and a browser window automatically opens, displaying the app’s user interface. Streamlit handles the backend server setup, frontend generation, and live interaction seamlessly, making deployment simple and effective.

**User Interface and Operation**

The Streamlit UI is designed for ease of use:

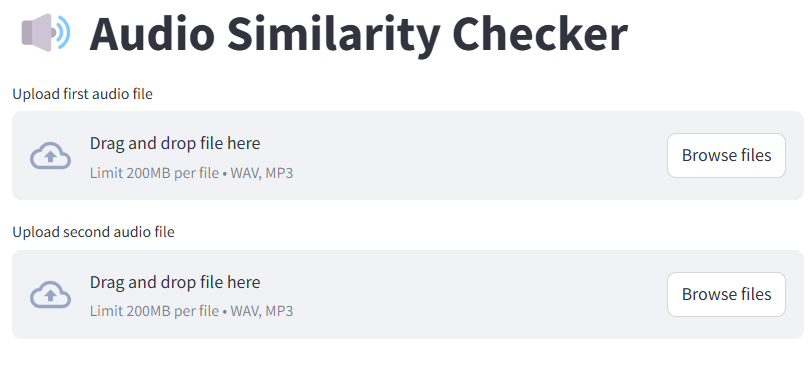
* **Audio Upload**: Users can upload two .wav audio files—one for each speaker they want to compare.
* **Processing**: Upon uploading, the app:
  1. Extracts speaker embeddings from both audio files using the pre-trained PyAnnote model.
  2. Computes the **cosine similarity** between these embeddings to measure how similar the voices are.
* **Output**: A similarity score is displayed on the screen, indicating the degree of match between the two speaker’s voices. A higher score implies greater similarity.

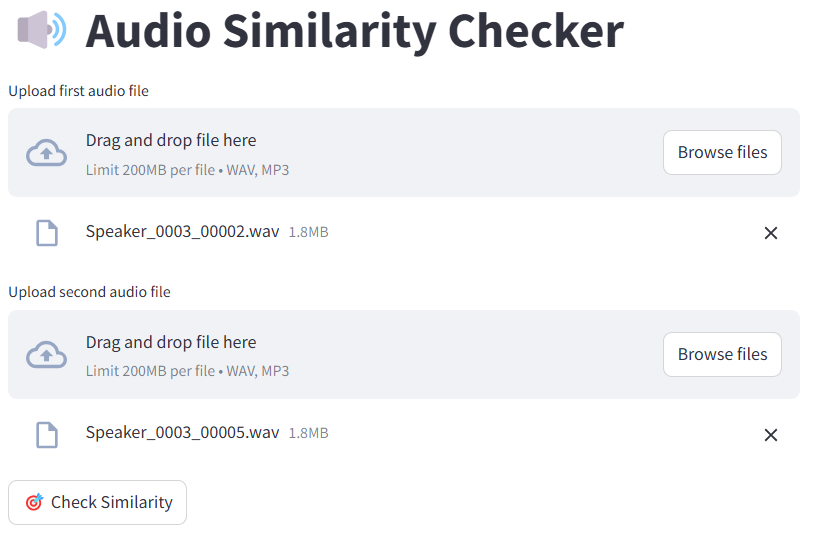
**System Workflow Summary**

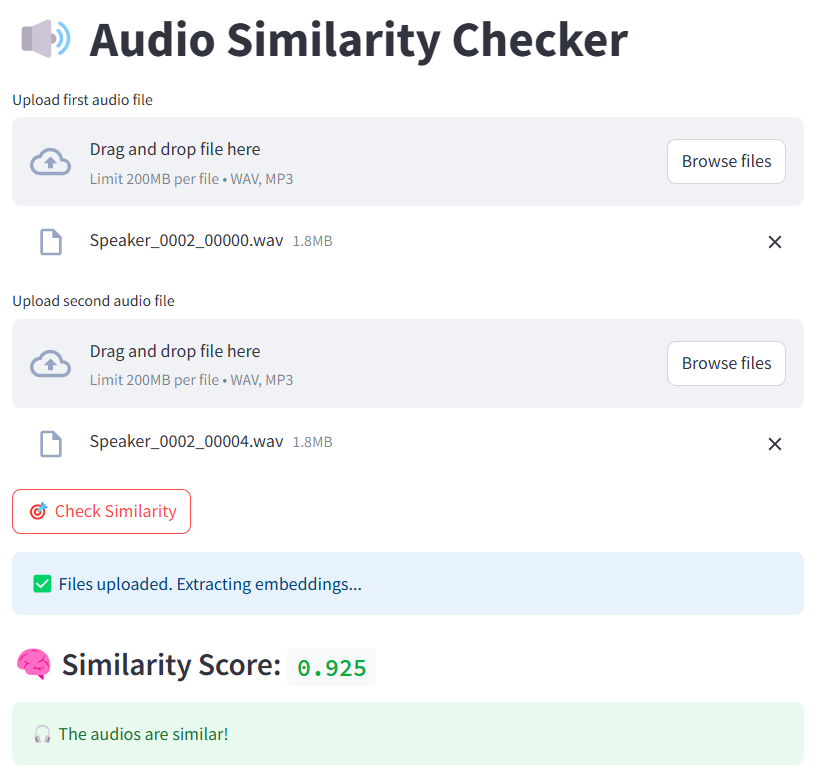
1. User uploads two audio files.
2. Each file is pre-processed and passed through a speaker embedding model.
3. Embeddings are compared using cosine similarity.
4. Result is shown on the Streamlit interface.

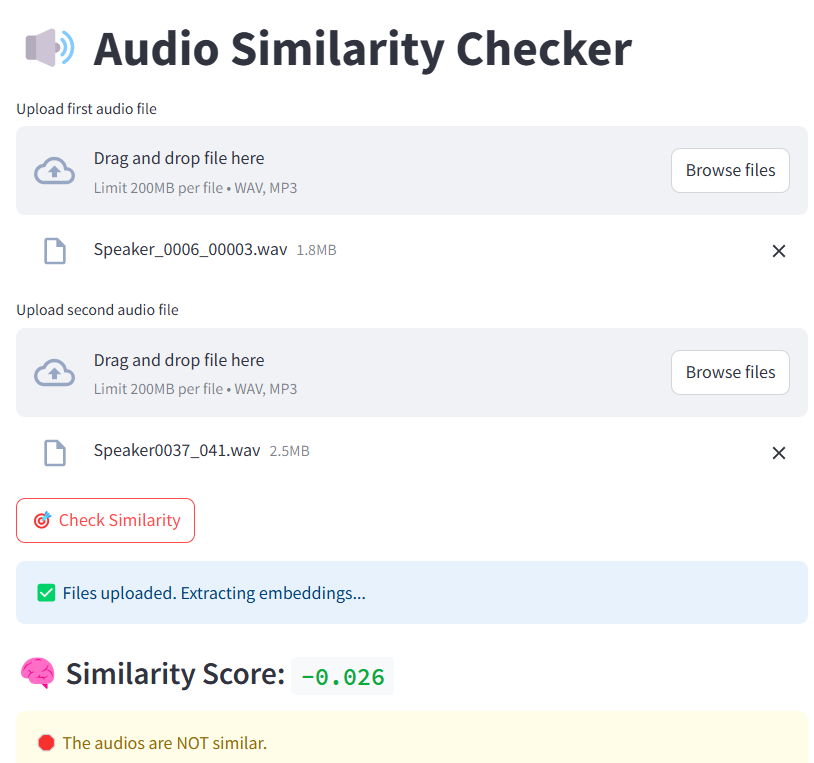
This deployment enables efficient testing and demonstration of the Cross-Lingual Voice Similarity system in a user-friendly manner without the need for complex infrastructure.

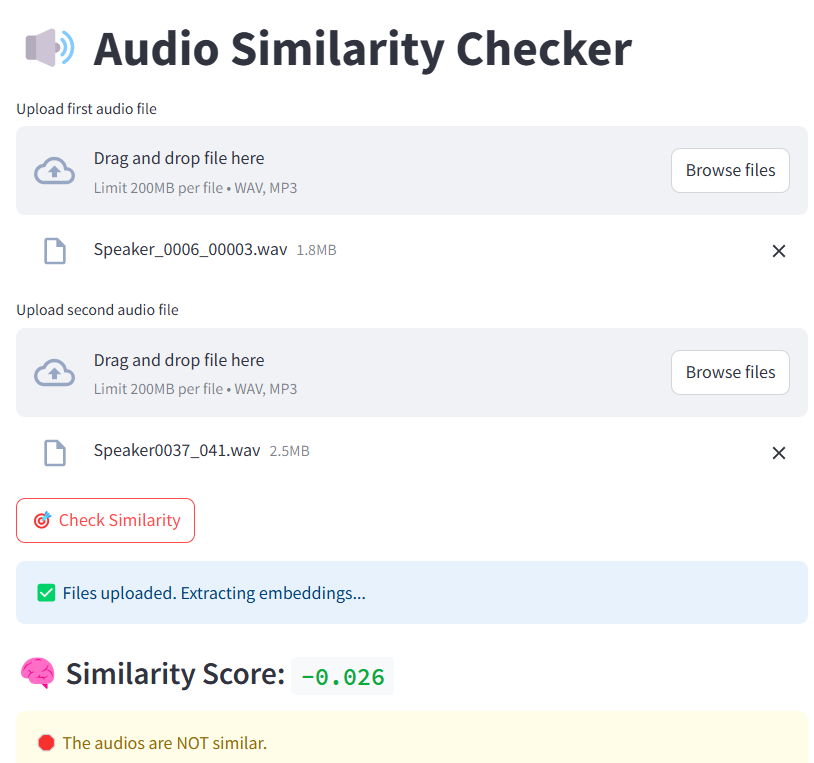
## Snapshot











## Limitations

Despite the effectiveness of the system, several limitations were observed during development and testing:

* **Sensitivity to Audio Quality**:
  + The model's accuracy significantly decreases when processing low-quality or compressed audio recordings. Clean, high-fidelity .wav files yield the most reliable similarity scores.
* **Background Noise Interference**:
  + The system is sensitive to background noise, which can distort speaker embeddings and negatively impact similarity results. Pre-processing techniques like noise reduction are not currently integrated.
* **Limited Language Support**:
  + This implementation is currently trained and tested only on English and Hindi. It may not perform accurately with other languages, especially tonal languages like Mandarin or Vietnamese, which present unique phonetic challenges.
* **Lack of Real-Time Capability**:
  + The system is not optimized for real-time or streaming audio processing. It works on pre-recorded .wav files and may exhibit latency if scaled to handle continuous input.
* **Cross-Lingual Generalization Constraints**:
  + While designed for cross-lingual speaker similarity, generalization across highly divergent languages may require more robust multilingual embeddings or fine-tuning on a larger and more diverse dataset.

## Future scope

To improve the scalability, flexibility, and real-world applicability of the system, the following enhancements are proposed:

* **Extend Language Support**:
  + Include more Indian and global languages such as Marathi, Tamil, Telugu, etc., to broaden the model’s multilingual capabilities and support diverse user bases.
* **Speaker Diarization Integration**:
  + Add speaker diarization to handle full-length conversations, enabling the system to segment and analyze individual speakers within multi-speaker audio files.
* **Audio Database and Similarity Search**:
  + Develop a searchable audio database where each audio file is stored along with its speaker embedding. This would allow the user to upload a single audio file and retrieve the most similar voices from the database, making the system suitable for applications like casting, voice matching, and speaker verification.
* **API Service for Industry Usage**:
  + Wrap the system into a RESTful API or microservice that can be integrated into industry applications—such as dubbing tools, call center analytics, or voice-based search engines.
* **Real-Time and Streaming Support**:
  + Upgrade the system to process streaming audio for real-time applications like live speaker verification, multilingual voice matching in video calls, or AI-based voice assistants.
* **Noise Robustness and Advanced Pre-processing**:
  + Implement noise filtering, silence removal, and voice activity detection (VAD) for better handling of field or call-quality recordings.
* **Deployment on Cloud with Scalable Backend**:
  + Host the system on platforms like AWS, Google Cloud, or Streamlit Cloud, and integrate a database backend (e.g., PostgreSQL, MongoDB) for scalable and remote access.

## Conclusion

This project explored the feasibility and practical implementation of cross-lingual speaker recognition using audio embeddings. The core objective was to determine whether speaker similarity could be measured reliably when voices are recorded in different languages—in this case, English and Hindi. By leveraging PyAnnote’s powerful pretrained speaker embedding models, we successfully extracted high-quality, language-agnostic representations of speaker voices.

These embeddings served as a foundation for comparing audio samples using cosine similarity, a straightforward yet effective metric for determining how closely two voice representations align in the embedding space. The simplicity of this approach, combined with its interpretability, makes it a strong starting point for multilingual speaker comparison tasks.

The development of an interactive Streamlit web application provided a user-friendly interface to upload audio files, process them, and display similarity scores in real-time. This not only made the system more accessible to non-technical users but also created a practical way to demonstrate the capabilities of cross-lingual voice comparison.

The results from initial testing are promising—especially given the challenges inherent to comparing voices across different linguistic, phonetic, and acoustic patterns. Despite current limitations such as noise sensitivity and language scope, the model showed the potential to retain speaker identity across languages, a task that is often considered difficult due to variations in pronunciation, prosody, and linguistic context.

With future improvements—including support for additional languages, noise reduction, real-time capabilities, and a searchable audio database—the system can evolve into a full-fledged solution for cross-lingual speaker recognition and retrieval. Such a system could be invaluable in domains like dubbing and media localization, security and forensics, call center analytics, voice assistants, and personalized AI systems.

In a multilingual world where voice is increasingly becoming a key interface between humans and machines, tools like this hold immense promise. This project demonstrates a meaningful and innovative step toward building technology that can understand, compare, and respect the uniqueness of human voices—regardless of the language they speak.

## References

* **PyAnnote-Audio: Speaker Embedding Models**  
  PyAnnote provides state-of-the-art speaker embeddings, originally designed for speaker diarization. Its pretrained models generalize well across different languages and environments, making it ideal for multilingual speaker similarity tasks.  
  <https://huggingface.co/pyannote/embedding>
* **Common Voice Dataset by Mozilla**  
  A multilingual, open-source dataset of voice recordings contributed by volunteers worldwide. Used for training and evaluating speech and speaker models.  
  <https://commonvoice.mozilla.org>
* **Librosa: Python Library for Audio Analysis**  
  A powerful Python package for music and audio analysis, providing features like spectrogram generation, audio loading, trimming, and more.  
  <https://librosa.org/>
* **Streamlit: Interactive Web Apps for Data Science**  
  An open-source Python framework for quickly building and deploying interactive web applications with minimal effort.  
  <https://streamlit.io>
* **Cosine Similarity – scikit-learn Documentation**  
  Cosine similarity is used to compute similarity between speaker embeddings in this project.  
  <https://scikit-learn.org>
* **VoxCeleb: Large-Scale Speaker Identification Dataset**  
  Nagrani, A. et al. (2017). VoxCeleb: A Large-Scale Speaker Identification Dataset. Demonstrates the effectiveness of large, diverse datasets for speaker verification and embedding learning.  
  <https://arxiv.org/abs/1706.08612>
* **Audio-Based Near-Duplicate Video Retrieval with Audio Similarity Learning**  
  Demonstrates how similarity learning can detect near-duplicate videos using audio embeddings, even with speed and pitch variations. Relevant for audio fingerprinting and retrieval.  
  <https://arxiv.org/abs/2010.08737>
* **Content-Based Representations of Audio Using Siamese Neural Networks**  
  Introduces a Siamese network to learn semantic audio embeddings for content-based retrieval. Validates the effectiveness of cosine and Euclidean similarity metrics.  
  <https://arxiv.org/abs/1710.10974>
* **EACeleb: An East Asian Language Speaking Celebrity Dataset**  
  A multilingual speaker dataset demonstrating how diarization and fine-tuning improve model accuracy across regional language groups.  
  <https://arxiv.org/abs/2203.05333>
* **Audio Similarity Detection (APSIPA 2024)**  
  Overview of audio similarity detection techniques and their applications, highlighting the role of deep learning and distance metrics in different similarity tasks.  
  <http://www.apsipa2024.org/files/papers/478.pdf>