



**Institute:** SRH University of Applied Sciences Berlin

**Degree program:** Master of Science

**Semester:** 4<sup>th</sup> Semester

**Course Name:** Computer Science (Artificial Intelligence and Big Data)

**Name of Professor:** 1) Prof. Dr. Alexander Iliev  
2) Prof. Dr. Vladimir Stantchev

**Title of work:** Speaker Recognition System  
with Advanced Security Functionalities

**Name of Author:** Suhas Yogeshwara

**Enrolment Number:** 3105758

**Email ID:** [3105758@stud-camus-berlin.de](mailto:3105758@stud-camus-berlin.de)

## Table of Contents:

Table of Contents:.....	2
Table of Figures:.....	5
List of Tables:.....	6
List of Code Snippets:.....	6
1. Acknowledgment:.....	8
2. Abstract:.....	9
3. Introduction:.....	10
4. Literature Review:.....	11
4.1. Recent Works:.....	12
5. Research Methodology:.....	15
5.1. Preprocessing:.....	16
5.1.1. Analog-to-Digital Conversion:.....	16
5.1.2. Short-Time Analysis:.....	19
5.2. Feature Extraction Techniques:.....	22
5.2.1. Mel-Frequency Cepstral Coefficients:.....	23
6. Early Speaker Recognition Approaches:.....	24
6.1. Gaussian Mixture Model:.....	24
6.1. Universal Background Model:.....	25
6.1. Support Vector Machines:.....	26
6. Deep Neural Networks:.....	27
6.1. Basic Parameters in Neural Network:.....	28
6.1. Recurrent Neural Network:.....	33
6.1.1. Parameter Efficiency:.....	34
6.1.2. Time Efficiency:.....	34
6.1.3. Robustness:.....	34
6.1.4. Feature Extraction:.....	34

6.2. LSTM (Long Short-Term Memory): .....	35
6.3. Attention Mechanism: .....	36
7. Speaker Recognition System threats and Limitations: .....	37
7.1. Imposter Attacks: .....	38
7.1.1. Feature-level analysis: .....	38
7.1.2. Model-Based approaches: .....	38
7.1.3. Fusion-Based Methods: .....	39
7.1.4. Deep Neural Networks: .....	39
7.2. Speaker Variability: .....	39
7.3. Privacy Concern: .....	40
7.3.1. Biometric Data Collection: .....	40
7.3.2. Voice Data Storage: .....	41
7.3.3. Audio Surveillance: .....	41
7.3.4. False Positive identification: .....	41
7.3.5. Discrimination: .....	42
7.4. Ethical considerations:.....	42
8. Market Survey on Speaker recognition system: .....	43
8.1. False acceptance and false rejection rate: .....	45
8.2. Compliance with regulations: .....	45
9. Datasets: .....	46
9.1. Recording Audio Signals:.....	47
9.2. Creating a Directory: .....	48
9.3. Benefits of Multi-lingual Datasets: .....	49
9.3.1. Data with 0.5 meters distance:.....	50
9.3.2. Data with 1 meter distance: .....	51
9.3.3. Audio Recordings with 1.5-meter distance: .....	52
9.3.4. Audio Recordings with 2 meters distance: .....	53

9.3.5. Audio Recordings with 2.5 meters distance: .....	54
9.3.6. Audio Recordings with minimal background noise: .....	56
10. Code Examples: .....	57
10.1. Feature Extraction (MFCC): .....	57
10.2. Padding all the feature extracted to same length: .....	58
10.3. Concatenation of all language files with respective speaker: .....	59
10.4. Data Augmentation Technique: .....	60
10.4.1. Speed Perturbation: .....	60
10.4.2. background Noise: .....	61
10.4.3. Room Simulation: .....	62
10.5. Addition of All the Speakers Files with data augmented files: .....	63
10.6. Combination of all the files of speakers: .....	64
10.7. Train, Validation and Test, Features and Labels: .....	65
10.8. GMM model using Sklearn library: .....	66
10.9. Support Vector Machine Code: .....	68
10.10. Recurrent Neural Networks for speaker recognition: .....	69
10.10.1. Multi-task learning RNN model: .....	72
10.10.2. Evaluating the Multi-task model: .....	75
10.10.3. Confusion Matrix visualization: .....	76
11. Testing on Raw Audio Data: .....	78
12. Contribution of this Work: .....	80
13. Future Scope: .....	80
14. Conclusion: .....	81
15. References: .....	83
16. Declaration by Candidate: .....	90

## Table of Figures:

Figure 1. Analog-to-Digital Converter.....	16
Figure 2. Sampled Signal. ....	17
Figure 3. 3-Bit Resolution with 8 – level. ....	19
Figure 4. Framed signal. ....	20
Figure 5. Hanning Window Sampled Signal. ....	21
Figure 6. Post-Processing Technique.....	22
Figure 7. Gaussian Distribution plot. ....	24
Figure 8. UBM plot of basic data points. ....	26
Figure 9. Basic Neural Network Structure. ....	27
Figure 10. Plot for Gradient Descent.....	31
Figure 11. Plot for Parameters vs Iterations. ....	32
Figure 12. Basic RNN Representation. ....	33
Figure 13. LSTM Mechanism used in RNN. ....	35
Figure 14. Attention weights of attention layer. ....	36
Figure 15. plot for rising demand in industry. ....	44
Figure 16. Audacity Software for Recording Audio Samples. ....	47
Figure 17. Directory creating and Storing the Audio recordings. ....	48
Figures 18. Proof of files stored in NPY format.....	59
Figures 19. Equal Error Rate and False Rejection Rate. ....	72
Figures 20. Confusion matrix labels.....	77
Figures 21. Complete Model Parameters. ....	77
Figures 22. The Testing Audio.....	78

## List of Tables:

Table 1: First Data with Zero Background Noise Recordings.....	49
Table 2: Easy Utterance dataset with minimal distance. ....	50
Table 3: Distance of 1 meter between speaker and recording device.....	51
Table 4: Distance of 1.5 meters with Zero Background Noise.....	52
Table 5: 2 Meters distance to speech signal recordings and system.....	53
Table 6: 2.5 Meters Distance to speech signal recordings and system.....	54
Table 7: 0.5 Distance from device to speaker with background noise. ....	56
Table 8: Training, validation, and testing set.....	66
Tables 9: Evaluation Metrics variations. ....	68
Table 10: Loss Evaluation table.....	75
Table 11: Accuracy Evaluation table. ....	75
Table 12: Testing Evaluation metrics. ....	76
Tables 13: Libraries and Software versions.....	79

## List of Code Snippets:

Code Example 1. Import Dependencies and Feature Extraction. ....	57
Code Example 2. Padding all the feature Extracted files to fixed length. ....	58
Code Example 3. Concatenation of Feature extracted files.....	59
Code Example 4. Speed Perturbation for audio signals. ....	60
Code Example 5. Addition of background noise to the speaker's voice. ....	61

Code Example 6. Room simulation code. ....	62
Code Example 7. Concatenate all the Speaker's files. ....	63
Code Example 8. One complete dataset of all speakers. ....	64
Code Example 9. Train, validation, and test sets. ....	65
Code Example 10. GMM model simple code for evaluating the dataset. ....	66
Code Example 11. Evaluation metrics with results. ....	66
Code Example 12. SVM for Speaker recognition system. ....	68
Code Example 13. Evaluation metrics for SVM. ....	68
Code Example 14. RNN using Keras Library. ....	69
Code Example 15. Model building and training.....	70
Code Example 16. Result with evaluation metrics for RNN.....	70
Code Example 17. Evaluation on test and other metrics. ....	71
Code Example 18. Multi-task model training.....	73
Code Example 19. Final model training. ....	73
Code Example 20. Result Evaluation. ....	74
Code Example 21. Testing multi-task learning model.....	75
Code Example 22. Confusion matrix.....	76

## **1. Acknowledgment:**

Prof. Dr. Alexander Iliev and Prof. Dr. Vladimir Stantchev, who oversaw my thesis, have my sincere gratitude for their invaluable advice, inspiration, and support throughout my research. Their knowledge, insights, and feedback have helped shape my ideas and improve the quality of my work.

I am also grateful to the faculty members of SRH University of Applied Sciences Berlin's Department of Computer Science (Artificial Intelligence and Big Data) for their stimulating discussions and constructive feedback, which have aided the development of my research.

In addition, I would like to thank my family and friends for their unwavering love, support, and understanding throughout this difficult but rewarding journey. Even when things got tough, their encouragement and motivation kept me going.

Conclude, I would like to extend my sincere gratitude to everyone who has helped this thesis come to completion in some way. No matter how small their contributions may have been, they have had a significant impact.

I appreciate you all.

Suhas Yogeshwara



## 2. Abstract:

Speaker Recognition is a technique that uses a specific person's voice commands to improve a system. Everyone has their own vocal folds and track. Individuals are recognized by their vocal characteristics, as well as differences in spoken language, accents, and dialects. In recent times, security issues such as imposter attacks and speaker variability on speaker recognition systems have had a significant impact on the system's performance. (Wang Q. , 2020). In this paper, we will discuss the current security issues with speaker recognition systems and how to overcome them.

The feature extraction module and the classification module are the two main parts of the speaker recognition system. Pitch, tempo, and spectral characteristics are some examples of pertinent information that is extracted from the audio signal by the feature extraction module. The classification module then makes use of this data to locate the speaker by comparing it to a database of speaker models that already exist.

Using a custom multi-language dataset, we evaluated our novel and improved speaker recognition system thoroughly. This improved system incorporates several advanced techniques, such as speaker and language attribute training, attention mechanisms and LSTM layers, and testing on the trained model on test dataset.

I conducted an in-depth ablation study to understand the impact of each innovation, systematically analysing the contribution of each component within our model. This investigation revealed the importance of each component in improving recognition performance.

**Keywords:** Speaker Recognition, Neural Networks, Gaussian Mixture Model, Support Vector Machine, Recurrent Neural Network, Speaker Variability, Speaker Imposter attacks, Feature Extraction Technique, Data Augmentation, Attention Mechanism, LSTM.

### **3. Introduction:**

The key features of recognizing a person by his or her voice, from communication to finding patterns for the speaker's emotions in his or her voice by the spoken words, are where technology has progressed thus far. The primary function of a speaker recognition system is to recognize a person's speech by using a trigger word and conducting a few simple commands. Personal speech recognition systems that not only listen to your voice but also issue commands for security-related decisions will be popular in the future. (Nilu Singh, 2017) Automatic speaker recognition systems have emerged as an essential means of validating identity in many e-commerce applications, as well as ordinary commercial transactions, forensics, and law enforcement (Mohammed algabri, 2017).

Speaker recognition technology has numerous and expanding applications. Control access to services such as voice dialing and voice mail, tele-banking, telephone shopping, database access related services, information services, security control for confidential information areas, forensic applications, and remote access to computers are all possible with this technique (Mohammed algabri, 2017). The speaker recognition system, the imposter attack and speak variability are the main security concerns. Attacks known as imposter attacks involve a third party trying to gain access to a system by pretending to be a legitimate user. Imposter attacks in speaker recognition systems happen when someone tries to impersonate a legitimate user's voice to gain access. Replay attacks, voice conversion, and synthetic speech are just a few of the techniques that can be used to commit imposter attacks.

The term "speaker variability" refers to the typical variation in a speaker's speech signal caused by a variety of factors, such as emotion, physical health, and environmental circumstances. How well speaker recognition systems function can be significantly impacted by speaker variability. This thesis discusses security issues and the development of an automatic speaker recognition system based on a neural network on different architectures and evaluating it for accuracy (Ashish Vaswani, 2017).

#### 4. Literature Review:

The first piece, titled "voice identification," appeared in 1962 and used a spectrogram rather than an oscillograph. Later, to cut costs and unreliability, computer algorithms took the place of human readers. The first algorithm, developed in 1970, used pattern matching. In the past, the costs associated with using human readings for Voice ID identification were too high, and they were known as voice print pros. Human readers are inconsistent and unreliable (Wang Q. , 2020).

The first algorithm ever created was "Pattern Matching", which took cues from human readings and used mathematical equations to measure the difference rather than performing a manual comparison. The failure to record voice signal variations was this strategy's only drawback. Later, statistical models for identifying differences in speech signals were developed; versions like the Gaussian Mixture Model, Universal Background Model, and Support Vector Machines were in use up until the time when Deep Learning overtook the New Age for Voice-ID Techniques. (Wang Q. , 2020) and most recently, there were the transformer models.

For many years, RNNs, GMM and SVM have been used to recognize speakers. These networks can detect temporal dependencies in speech signals and can deal with speaker variability. Some models based on LSTM or GRU architectures have recently proven to be effective in speaker recognition tasks. The authors propose using deep neural networks to learn features that are optimized for the task of speaker recognition automatically (Snyder, 2020). Attention-based models have been proposed in speaker recognition to focus on speaker-specific features while filtering out irrelevant information. These models have been demonstrated to be effective in dealing with speaker variability, background noise, and channel variations (Jung, 2019).

Transformer-based models have recently been proposed for speaker recognition tasks. These models outperformed traditional models in terms of capturing long-term dependencies in speech signals (Kong, 2019). The traditional and recurrent neural networks will be used in my paper, and I will discuss how to use the custom dataset to reduce the effect of speaker variability and imposter attacks.

#### 4.1. Recent Works:

**1. Federated learning for privacy-preserving speaker recognition systems** by ABRAHAM WOUBIE, AND TOM BÄCKSTRÖM. Human speech samples may contain valuable information that they do not want to share with other people. Using edge devices where the data is stored, federated learning makes it possible to train a shared model without disclosing any confidential information (ABRAHAM WOUBIE, 2021).

**2. Deep Speaker Recognition, process, and Challenges** by ABU QUWSAR OHI, M. F. MRIDHA, MUHAMMAD MOSTAFA MONOWAR. The current deep learning networks being used to build speaker recognition systems are discussed in this paper, along with their methods and difficulties compared to more conventional models. How neural networks are influenced to perform a speaker recognition system. Convolutional neural networks, recurrent neural networks, and attention mechanisms are just a few examples (ABU QUWSAR OHI, 2021).

**3. Bias in Automated Speaker Recognition** by WIEBKE TOUSSAINT, AARON YI DING. The systematic study of bias in automated speaker recognition is lacking. We present a comprehensive empirical and analytical analysis of bias in speaker verification, a voice biometric and essential task in automated speaker recognition. With the use of Voxceleb1 Dataset. The dataset contains speech recordings of thousands of celebrities and public figures culled from a variety of sources, including interviews, speeches, and podcasts. In various research works, the VoxCeleb dataset has become a standard benchmark for evaluating the performance of speaker recognition systems (WIEBKE TOUSSAINT, 2022).

**4. State of art Speaker Recognition System** by Marcos Faundez-Zanuy, Enric Monte-Moreno. The authors of this paper discuss why, despite technological advancement, the development of Speaker recognition has lagged and is still being researched. also discusses the advantages and disadvantages of using neural networks. Neural networks can achieve high accuracy in speaker recognition tasks, especially when trained on substantial amounts of data. can withstand variations in speech signals such as noise, accent, and different recording conditions. difficult to understand, making it difficult to diagnose and fix system problems. (Marcos Faundez-Zanuy, 2022).

5. A subset of deep learning models called transformers are identified by specific architectural characteristics. They were first introduced by Google researchers in the now-famous "**Attention is All you Need**" paper (Ashish Vaswani, 2017). The encoder decoder models, which had only recently begun to gain popularity two or three years earlier, are specifically used in the Transformer architecture. However, up until that point, these models, which relied on LSTM (Long Short-Term Memory) and other RNN (Recurrent Neural Networks) variations, only used attention as one of their mechanisms have been overcome with transformer models for better performance and efficiency (Amatriain, 2023).

6. **Attention-Based Models for Speaker Recognition** by Okko Räsänen, Tuomas Virtanen, and Moncef Gabbouj The authors suggested an attention-based speaker recognition model that employs a self-attention mechanism to concentrate on the input signal's most illuminating elements. The system is made up of two components: a feature extraction module that takes the audio signal's Mel-frequency cepstral coefficients (MFCCs) and an attention-based model that creates a fixed-length representation of the input signal using a self-attention mechanism enables the model to concentrate on the most informative parts of the audio signal, improving its ability to distinguish between different speakers. Furthermore, using MFCCs as input features can help to reduce the dimensionality of the input signal and improve the model's efficiency (Okko Räsänen, 2020).

7. The authors of "**Speaker Recognition with Transformer**" propose a novel framework for speaker recognition that employs transformer models for feature extraction and classification. The transformer models have been used successfully in natural language processing tasks and have shown a lot of promise in modelling sequential data. The authors modified the transformer architecture to process speech signals in the form of Mel-frequency cepstral coefficients (MFCCs) in this paper (Le Zhang, 2017).

The proposed framework takes advantage of the transformer model's strengths in capturing sequential dependencies and adapts it to process speech data via MFCCs. This enables the model to learn speaker-specific patterns from input speech signals and to make accurate predictions on speaker recognition tasks (Le Zhang, 2017).

**8. The authors of "Self-Attentive Residual Learning for Speaker Verification"** propose a framework for self-attentive residual learning for speaker verification. The proposed method for feature extraction and classification uses a transformer-based model and achieves competitive results on the VoxCeleb1 and VoxCeleb2 datasets. The proposed framework is made up of two modules: feature extraction and classification (Zhang, 2021).

The feature extraction module is intended to process input speech signals and extract speaker-specific features. It is made up of a residual self-attentive block, which is a combination of self-attention layers and residual connections. Self-attention layers enable the model to concentrate on critical parts of the input sequence, capturing long-term dependencies between speech frames. Residual connections improve gradient flow during training, reducing the vanishing gradient problem and promoting better optimization (Zhang, 2021).

The feature extraction module's extracted features are mapped to speaker embeddings by the classification module. Speaker embeddings are low-dimensional representations of speaker identities that allow for easier speaker comparison and verification. The classification module is typically made up of fully connected layers that map features to speaker embeddings (Zhang, 2021).

**9. "Attention-Based End-to-End Speaker Recognition Using Transformer Encoder"** proposes an end-to-end speaker recognition framework based on feature extraction and classification using a transformer encoder. On the VoxCeleb1 and VoxCeleb2 datasets, the proposed method achieved state-of-the-art performance. A transformer encoder, an attention-based aggregation module, and a classification layer are included in the proposed framework. The transformer encoder takes in the raw audio waveform and generates a series of hidden representations. The hidden representations are aggregated into a fixed-size embedding that captures the speaker information by the attention-based aggregation module. Finally, the classification layer associates speaker embeddings with speaker identities (Xue, 2021). The proposed approach was evaluated on the VoxCeleb1 and VoxCeleb2 datasets, and the results showed that it outperformed previous state-of-the-art methods by a significant margin. The authors also conducted ablation studies to assess the efficacy of various components of their proposed framework.

## 5. Research Methodology:

In the step-by-step process of developing a robust speaker recognition system, Analog-to-Digital converter preprocessing, feature extraction, early Models used in building a system, Latest Deep Learning Approaches, Speaker recognition Major threats and how to overcome them, last but not the least GMM, SVM and Recurrent Neural Networks for the custom dataset in hand for the robust Speaker recognition System, but first I would like to point out a few things, such as the What are speaker recognition system's limitations, which are imposter attacks and speaker variability, why do we need them. The reason for creating a new system is with all the advancements in this domain. Major threats which limit effective and better performance.

The most recent AI language model (Transformers, BERT, GPT-3, etc.) has significantly improved and offers better efficiency with the dataset provided because it accepts both structured and unstructured dataset. to discover the pattern and produce the results. Systems for identifying speakers are based on RNN. long short-term memory, attention, and attention mechanism of recurrent neural networks. Therefore, why not create a model that not only recognizes the owner but also recognizes imposter attacks? The dataset provided in this paper is multilingual (6 speakers, seventy utterances each), which works better for variability as each language has different dialects, accents, pitches, tones, and other linguistic features.

The dataset consists of 6 speakers and 420 utterances in total, with recordings varying in length (0.5 meters, 1 meter, 1.5 meters, 2 meters, 2.5 meters) between the device and the speaker and in various environments where the system attempts to learn the dependencies and trains with the data for hidden patterns and to have consecutive relations between the data points.

My improved approach incorporates attention mechanisms, allowing the model to focus on relevant segments within audio sequences and thus improve discriminative power, which is critical for variable-length sequences such as spoken language.

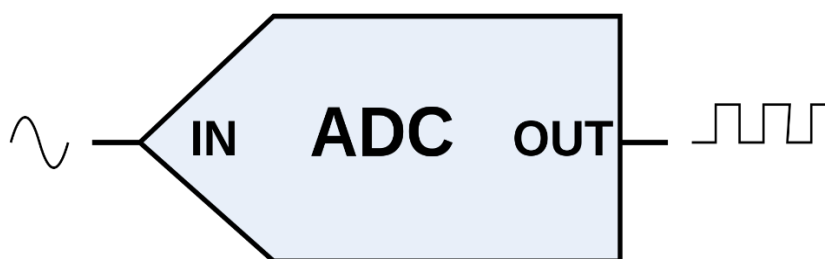
By including both speaker and language information, your model can capture a more complete representation of the audio data. This can improve its ability to differentiate between speakers and languages, as well as detect spoofing attempts.

Furthermore, integrating LSTM layers captures temporal dependencies in sequential data, significantly improving performance, especially in the context of speaker recognition, where speech is inherently sequential.

## **5.1. Preprocessing:**

### **5.1.1. Analog-to-Digital Conversion:**

Computers can only comprehend digital data streams made up of strings of zeros and ones. which is a pre-processing technique that involves converting a continuous analog signal to a discrete digital signal. Other applications involve two-way conversion. mathematical methods like waveform, the Fourier Transform, and the Fast Fourier Transform (FFT). These are the methods for representing an audio signal in a discrete format without losing any of the original signal's information. The fundamental conversion between an input Analog signal and an output digital signal is shown in the diagram below (Arzaghi, 2020).



**Figure 1. Analog-to-Digital Converter.**



#### 5.1.1.1. Sampling:

According to Nyquist-Shannon Theorem – The highest frequency component in audio signals is typically the Nyquist frequency, which is half the sampling rate. Avoid aliasing and accurately reconstructing the original audio signal, the sampling rate should be at least twice the Nyquist frequency. For example, because the human ear can hear frequencies up to 20 kHz, the sampling rate for an audio signal containing these frequencies should be at least 40 kHz. This is why the sampling rate of standard CD quality audio is 44.1 kHz (Shannon–Nyquist–Kotelnikov–Whittaker, 2020).

The original Analog signal can be perfectly recovered from the discrete values created by sampling if a system samples an analog signal uniformly at a rate that exceeds the signal's highest frequency by at least a factor of two (Shannon–Nyquist–Kotelnikov–Whittaker, 2020).

Formula:  $f(\text{Sampling}) = 2 * f(\text{input}_{\text{signal}})$

Example: if the input signal is of frequency 8000 Hertz, then the sampling frequency would be 16000 hertz i.e., 16000 sample points per second.

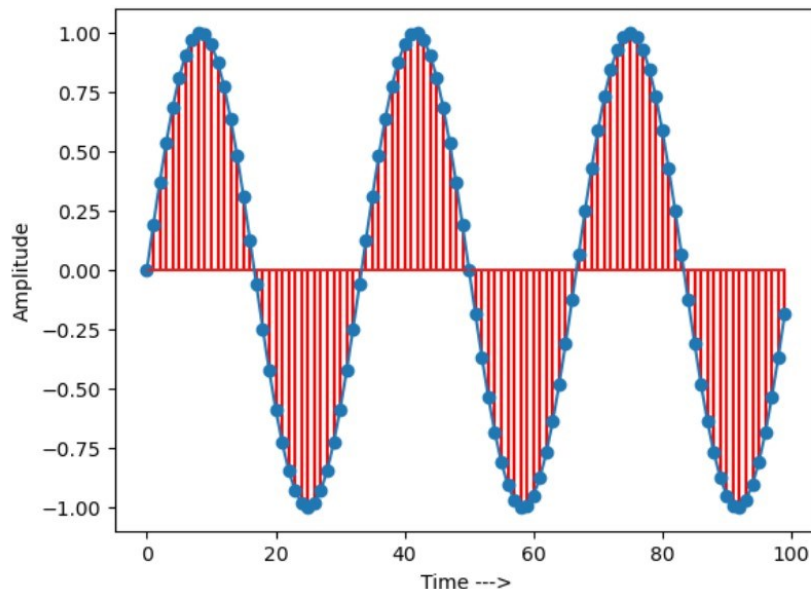


Figure 2. Sampled Signal.

**Explanation:**

The diagram above was created with librosa and the NumPy library at a sampling rate of 44100 Hz. The audio sample ('english\_0.5D0BGNSUHAS.wav') was taken from the datasets. The sample signal was plotted and displayed using a sine wave. The plot was created with the matplotlib library, with time as the x-axis, amplitude as the y-axis, and sampled points of 100 (to properly visualize the plot).

When discussing audio and speech technologies, this process is critical. acknowledge to everyone who contributed to making the implementation of sampling to a raw audio signal less difficult.

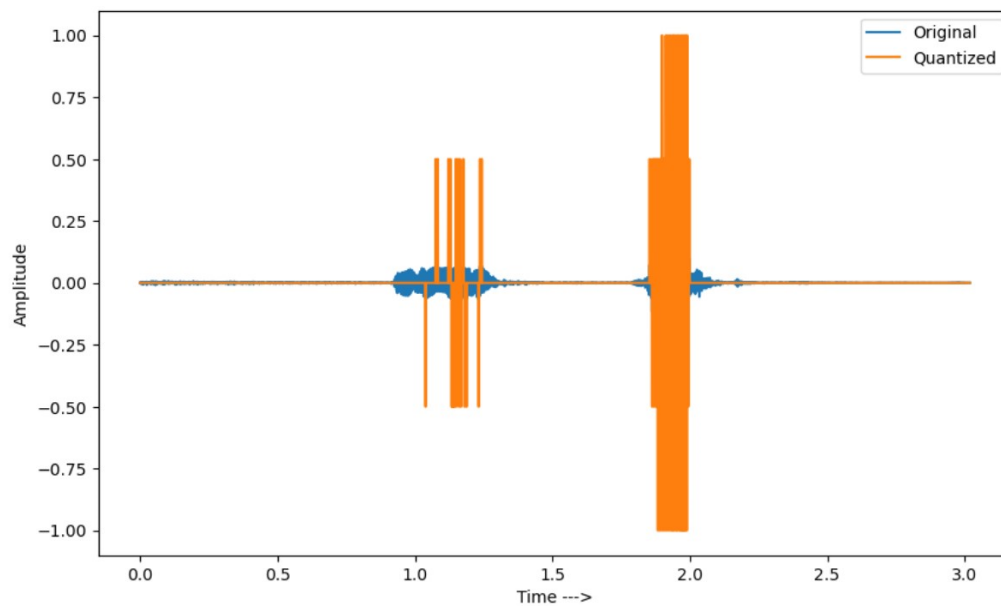
**5.1.1.2. Quantization:**

The process of converting continuous analog signals, such as an audio waveform, into a discrete digital representation is known as quantization. Quantization is the process of mapping the amplitude of each sample of an analog audio signal to a corresponding digital value that can be represented using a finite number of bits in the context of audio processing. The continuous amplitude values of an analog audio signal are divided into a finite number of discrete levels during quantization. The bit depth of the quantization determines the number of levels. A 16-bit quantization scheme, for example, can represent the amplitude of each sample using 216 (or 65,536) discrete levels, whereas an 8-bit quantization scheme can only represent the amplitude with 28 (or 256) levels (Wang Q. , 2020).

Quantization error is introduced during the quantization process, which is the difference between the true analog value and the nearest quantized value. This quantization error can degrade signal quality, especially for low-level signals or signals with a wide dynamic range. It is critical to use a high enough bit depth during quantization to minimize the effects of quantization error. In practice, audio signal quantization is typically done during the analog-to-digital conversion (ADC) process. The ADC converts the continuous analog audio signal into a digital signal that a computer or other digital device can process (Wang Q. , 2020).

The analog signal is sampled at regular intervals, and each sample's amplitude is quantized to a digital value. Using digital audio processing techniques, the resulting digital audio signal can be stored, transmitted, and processed (Wang Q. , 2020).

Other factors that can affect the quality of digital audio recordings, in addition to bit depth, include sampling rate, number of channels (mono or stereo), and encoding format (e.g., MP3, WAV, FLAC). These factors interact in complex ways, and the best settings are determined by the specific application and the user's priorities (Wang Q. , 2020).



**Figure 3. 3-Bit Resolution with 8 – level.**

### **Explanation:**

The plot above depicts the quantized and original signal with a bit depth of 4 and a maximum amplitude of 1.0, and it makes use of the `sound_file` library, which is one of the best Python packages for manipulating audio signals.

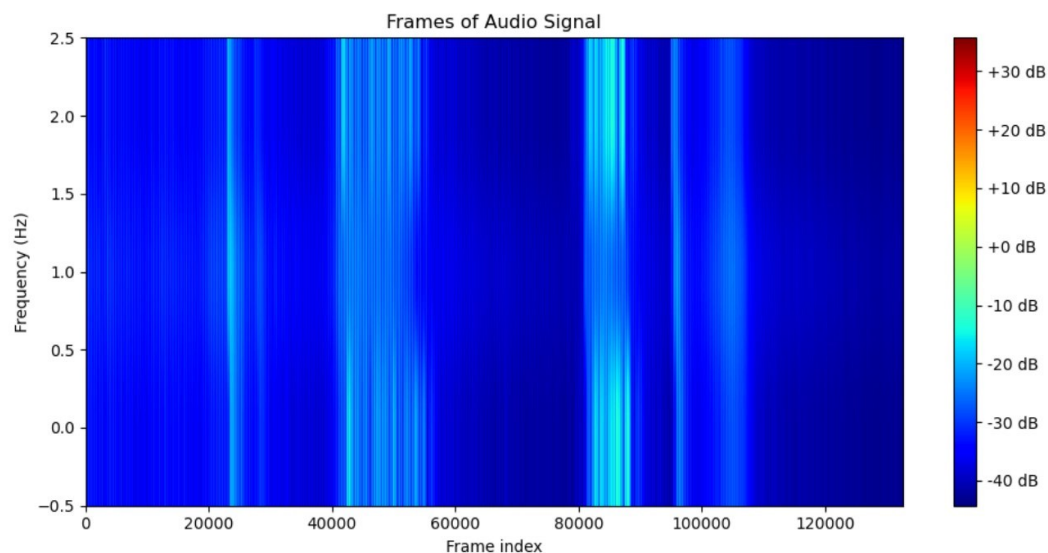
### **5.1.2. Short-Time Analysis:**

Short-time audio signal analysis involves splitting the signal into small, overlapping frames and analysing each frame separately. This method is useful for capturing time varying signal characteristics such as changes in pitch, loudness, and spectral content. Frame the signal, Pre-emphasis, Windowing, Fourier transform, Mel-frequency scaling,

Log-spectral coefficients, Delta coefficients are all steps in calculating short-time analysis for audio signals. Join the features, Normalization, and Classification together. Machine learning algorithms such as support vector machines, decision trees, random forests, and deep neural networks are commonly used for classification. It is frequently used in conjunction with other signal processing techniques, such as filtering, noise reduction, and feature normalization, to improve the analysis's quality and robustness. In subsequent sections, we will go over framing, window function, and other traditional models (Wang D. L., 2018).

#### 5.1.2.1. Framing:

Framing is the process of dividing an audio signal into overlapping frames of equal length. Typically, a sliding window technique is used, in which a fixed-size window is slid along the signal with a specified overlap between adjacent frames. Frames are typically extracted with 50% or less overlap between adjacent frames. Overlapping frames help to maintain frame continuity and reduce spectral artifacts at frame boundaries. (Kamil A. Kamiński, 2022).



**Figure 4. Framed signal.**

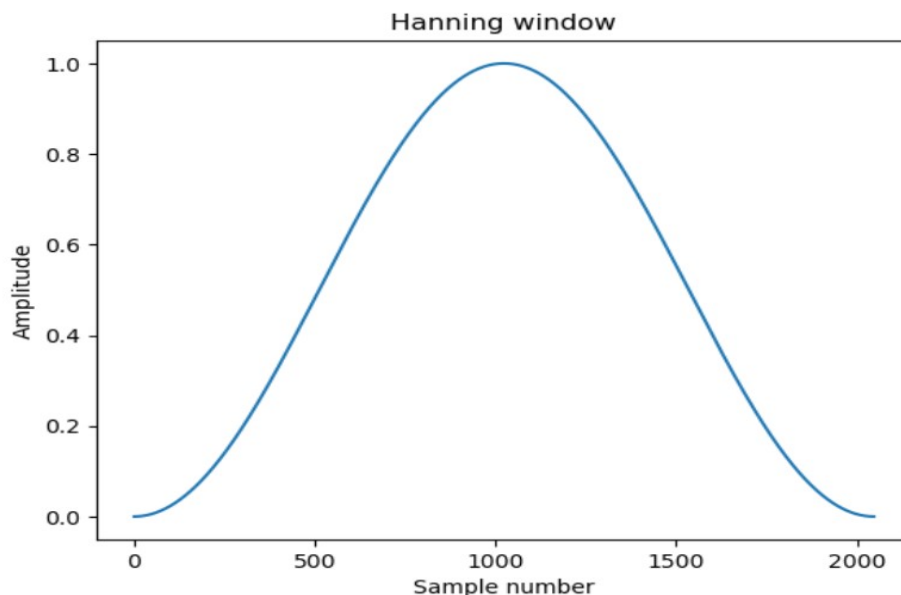
### Explanation:

This is the 5 seconds and a sampling rate of 16 kHz. If we want to divide the signal into 20-millisecond frames with 50% overlap, we must first determine the number of samples per frame as:  $\text{sampling\_rate} * \text{frame\_duration} = 16000 * 0.02 = 320$ . The framed audio sample with the file name **english\_0.5D0BGNSUHAS.wav** and a frame size of 20ms was visualized using librosa and matplotlib.

#### 5.1.2.2. Window Function:

A window function is a mathematical function that is used in signal processing to modify a signal by multiplying it by a windowing function. The goal of using a window function is to reduce the effect of signal leakage, which occurs when a signal's edges are abruptly cut off, resulting in signal distortion (Wang Q. , 2020).

The Hamming window, which is a smooth function that gradually reduces the amplitude of the signal towards the edges, is a commonly used window function in the case of speaker recognition. The Hamming window is a cosine-shaped window function that is used to taper the signal's edges to zero, reducing signal leakage and improving the accuracy of speaker recognition algorithms (wikipedia, 2014).



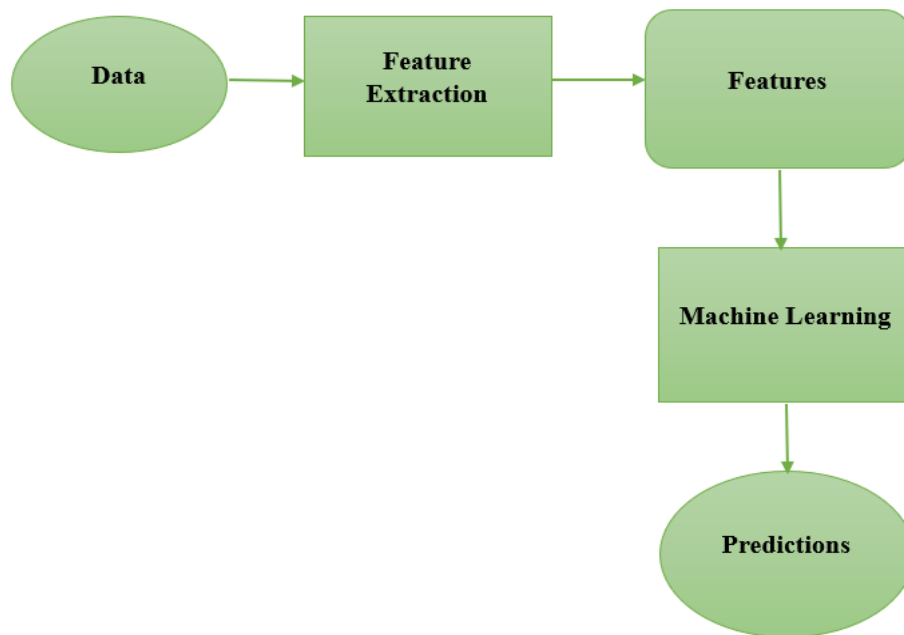
**Figure 5.Hanning Window Sampled Signal.**

### **Explanation:**

The above Signal diagram explains the Hanning window function following the gaussian distribution. The edges of the frames are adjusted to maintain the spectral leakage limit of the segment signal.

## **5.2. Feature Extraction Techniques:**

Under noisy environmental conditions, feature extraction in speaker recognition systems is a challenging task. We proposed a speaker recognition MFCC feature to increase the feature's robustness. Then, to improve feature discrimination, we extracted speech characteristics based on Mel-scale filter banks. A good feature should typically, in general, be independent, discriminative, and informative (Jiang Lin, 2020) (Wang Q. , 2020).



**Figure 6. Post-Processing Technique.**

### **Explanation:**

The flow chart served as the foundation for all machine learning projects, upon which models will be built and tested, and with multiple testing mechanisms.

### **5.2.1. Mel-Frequency Cepstral Coefficients:**

The frequently employed feature extraction method for speaker recognition systems is called MFCC, which compares the frequency of the audio signal to Mel-frequency bands before extracting features in accordance with the results. There are numerous additional methods. The crossing rate of zero, power-normalized Cepstral coefficients, such as spectral centroid and spectral roll-off, are like MFCC with the exception that gamma-filtered banks will be used for the analysis (Wang Q. , 2020).

#### **5.2.1.1. Pre-Processing Technique:**

Analog-to-digital converter followed by short-time analysis the discrete Fourier transform a mathematical technique is applied to the signal segments. where the time domain is converted into frequency domain components.

Formula:  $CC(n) = \sum E_k * \cos(n * (k - 0.5) * \pi / 40)$

#### **5.2.1.2. Mel-Filter Banks:**

Band pass- filter where the total frequency components are divided into overlapping frequency bands of the audio clip. Then the signal is comprised to a logarithmic scale. The high-pass filter is used where the higher frequencies will be allowed, which in turn increases the overall Signal-to-Noise Ratio (SNR). If increased SNR means, the signal is of excellent quality. Next step is to use Inverse Discrete Fourier Transform on the signal to obtain the time-frequency features (Wang Q. , 2020).

#### **5.2.1.3. Delta and Energy Representation:**

These are Additional Features for the better signal quality. Delta is the representation of Delta Banks where the rate of change of MFCC's is observed over time. Energy is Measure of overall energy or power in the signal (Wang Q. , 2020).

## 6. Early Speaker Recognition Approaches:

Gaussian Mixture Model (GMM), Universal Background Model (UBM), and support vector machines were early speaker recognition models (SVM). All these models were in use up until the beginning of 2007, when Deep Neural Network Architecture took their place as the foundation for creating a reliable and secure speaker recognition system.

### 6.1. Gaussian Mixture Model:

A common technique for identifying speakers is the Gaussian mixture model (GMM), which models the probability distribution of speech features using a combination of Gaussian distributions. The speaker identity is established by identifying the speaker who has the highest probability of producing the input speech features. In GMM-based speaker recognition, the GMM is trained on a set of speech utterances from each speaker. (Kamil A. Kamiński, 2022).

The ability of GMMs to represent complex data distributions and capture correlations between different variables is one of their main advantages. They are also adaptable and can be used for a variety of purposes. They can, however, be computationally expensive to train and may necessitate careful tuning of hyperparameters like the number of mixture components (Wang Q. , 2020).

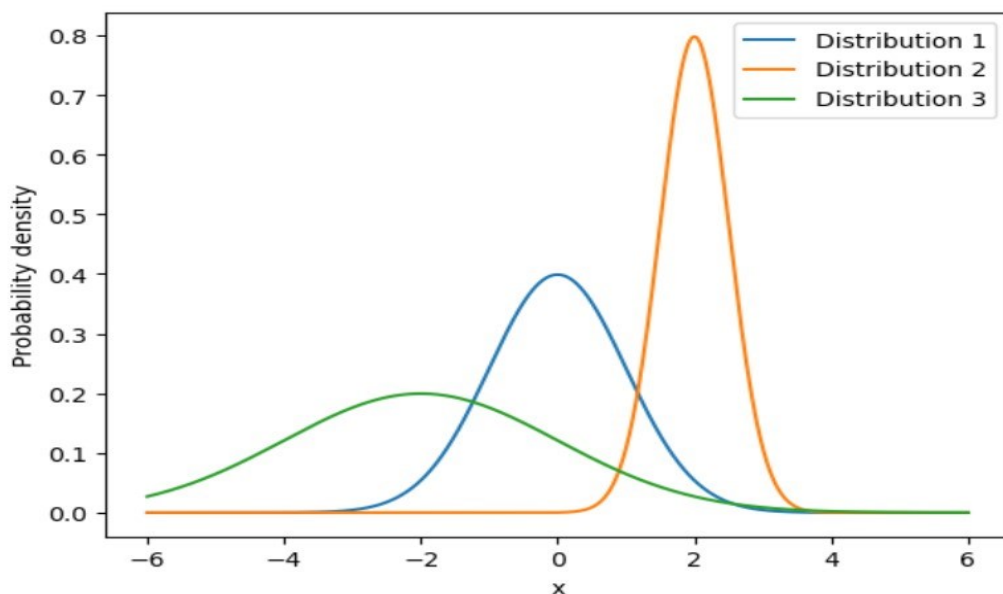


Figure 7. Gaussian Distribution plot.



**Explanation:**

The resulting plot will show the shape of each Gaussian distribution, with the mean parameter corresponding to the peak of each distribution and the standard deviation parameter corresponding to the width of each distribution. The data points for each gaussian distribution were 0,1,2,0.5, -2, 2. Which are the mean and variances of the distributions.

Formula: Probability Density Function (Wang Q. , 2020).

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x-\mu}{\sigma}}$$

Where:  $\sigma$  = standard Deviation

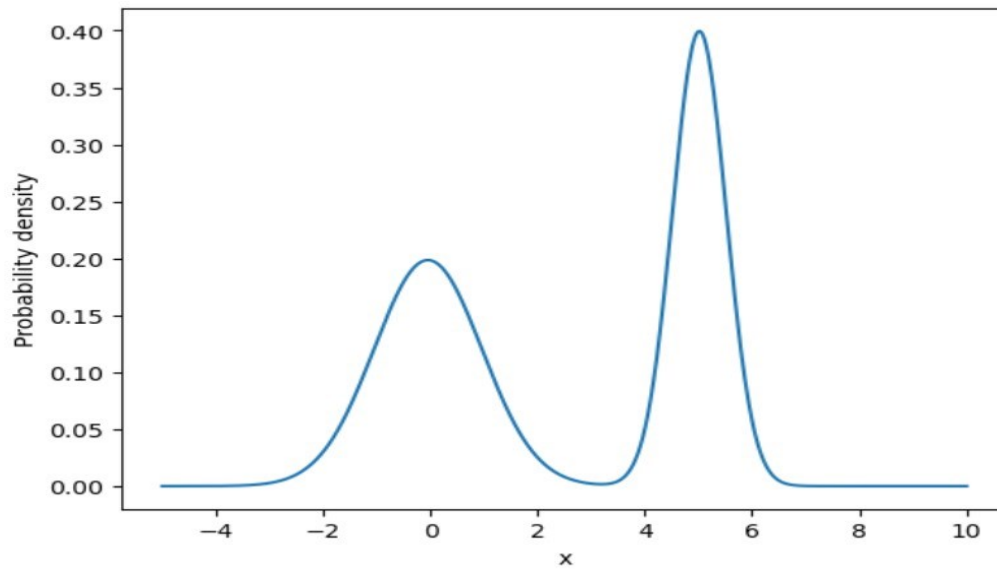
$\sigma^2$  = variance

$\mu$  = Mean

**6.1. Universal Background Model:**

A statistical model known as the Universal Background Model (UBM) is used in speaker verification systems to represent the distribution of speech acoustic features uttered by any potential speaker. Assess whether the voice of an unidentified speaker is sufficiently like that of the claimed speaker, the UBM is used as a reference model to compare against the speech features of an unidentified speaker (Gunawan, 2018).

Issues with GMM When we need an open set, it assumes closed set speaker identification, but there are still many issues, such as imposter attacks (Wang Q. , 2020). A GMM serves as the universal background model. The standard expectation maximization algorithm can estimate parameters. Due to UBM's speaker independence and straightforward training data pooling solution, there are no imposter data at the training data (Kamil A. Kamiński, 2022).



**Figure 8. UBM plot of basic data points.**

#### **Explanation:**

The UBM is a smoothed estimate of the sample data's underlying probability distribution. The plot's x-axis displays the values of the input variable, which is a univariate scalar in this case. The probability density for each value of the input variable is shown on the y-axis. A Gaussian mixture model (GMM) with two components is used to estimate the probability density.

#### **6.1. Support Vector Machines:**

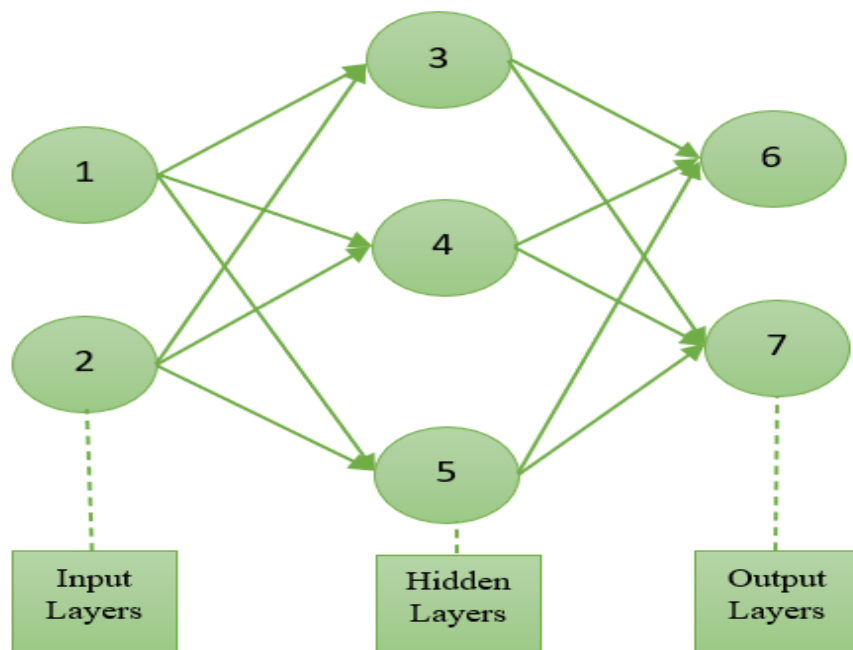
Systems for speaker recognition can make use of support vector machines (SVMs), a type of supervised machine learning algorithm. Speaker recognition technology is used to recognize people by their distinctive vocal traits. This can be done either through speaker identification or speaker verification, which involves determining whether the voice of a claimed speaker matches their recorded voice (i.e., determining the identity of an unknown speaker based on their voice) (Ganapathy, 2014).

SVMs can manage high-dimensional data and effectively separate classes in the data space, making them especially helpful for speaker recognition. Mel-frequency cepstral coefficients (MFCCs), which capture the spectral characteristics of speech signals, are a common technique for extracting the input features for speaker recognition from speech signals (Ganapathy, 2014) (Wang Q. , 2020).

SVM performance is affected by hyperparameter selection, such as kernel type, kernel width, regularization parameter, and so on. This can be difficult in speaker recognition, where determining the best hyperparameters may necessitate extensive tuning and cross-validation. SVM may be unable to find an optimal hyperplane that separates the different speaker classes when the data is non-linear. SVM models can be challenging to interpret, making it difficult to determine which features are most important for classification. These are some of the limitations of the speaker recognition system (Ganapathy, 2014).

## 6. Deep Neural Networks:

An artificial neural network that has an input layer, an output layer, and a hidden layer is called a deep neural network. The human brain neuron's dendrites, axon, and cell body, as well as its ability to transmit and receive electrical signals, serve as the foundation for the neural network's construction. Neural networks were created with the same design (Wang Q. , 2020).



**Figure 9. Basic Neural Network Structure.**

**Explanation:**

The plot was designed to use matplotlib and NumPy library with 2 input nodes, 3 hidden layer nodes and 2 output layer nodes.

The formula and the basic structure of neural network is shown in the next section.

Formula:  $h(x) = \sigma(w^T x + b)$

Where:  $w$  = weights or channels that are used as the connecting medium between neurons of layers.  $X$  = input signal and  $B$  = Bias is a Every Neuron. Weights and Bias are the Parameters.

**6.1. Basic Parameters in Neural Network:**

Here we investigate the basic parameters which are required in building a Neural Network Architecture where these components play a vital role in developing a customized Speaker recognition System.

**Weights:** Weights are parameters in a neural network that adjust the strength of connections between neurons. Each connection between two neurons is assigned a weight, which determines how strongly the first neuron's output influences the second neuron's input. Before training, the weights in a neural network are typically initialized at random. The network adjusts the weights during training to improve its performance on the training dataset. After training, the weights are fixed, and the network can make predictions on new input data (Wang Q. , 2020).

**Bias:** Bias is an additional parameter that is added to each neuron's input before it is passed through the activation function. The bias term allows the neuron to shift its activation function left or right, changing the range of input values that will cause the neuron to fire. It is significant because it enables the neural network to model more complex relationships between inputs and outputs. Without bias, the activation functions of the neurons would all be centred at zero, and the network would be unable to model relationships that require the activation functions to be shifted (Johnson, 2020).

**Hidden Layers:** The hidden layer oversees creating a set of intermediate representations of the input data, which the output layer can then use to make predictions or decisions. Each neuron takes the output of all the neurons in the previous layer as input and applies an activation function to this input to compute its output. The weights and biases associated with each neuron determine the strength of its connections to the previous layer and are learned during the training process. The number of hidden layers and neurons in each hidden layer are hyperparameters that can be tuned to improve network performance (Wang Q. , 2020).

**Activation Function:** it is a mathematical function that is applied to each neuron's output in the network. The activation function's purpose is to introduce nonlinearity into the neuron's output, allowing the network to model complex nonlinear relationships between the input and output data. A neuron's output can be represented as a linear combination of its inputs, weighted by the strength of the connections between the neurons. This output would be a simple linear function of the inputs without an activation function, which is not flexible enough to model many real-world problems (Wang Q. , 2020).

**Sigmoid Function**, hyperbolic tangent function, and rectified linear unit (**ReLU**) function are all common activation functions in neural networks. The sigmoid and hyperbolic tangent functions are S-shaped smooth curves with outputs in the ranges  $[0, 1]$  and  $[-1, 1]$ , respectively. The ReLU function is a straightforward threshold function that returns the input if it is positive and zero if it is negative (Wang Q. , 2020). The choice of activation function can have a significant impact on neural network performance. The sigmoid and hyperbolic tangent functions, for example, can suffer from the problem of vanishing gradients, in which the gradients become small in the curve's tails, making deep neural network training difficult. The ReLU function is not affected by this issue, but it is susceptible to "dying ReLU" problems, in which some neurons become permanently inactive during training and produce zero outputs (Wang Q. , 2020).

**Back Propagation:** The backpropagation algorithm computes the output by first passing the input through the network. The output is compared to the true output (i.e., the target output), and the difference is calculated as the error. The error is then propagated backwards through the network, beginning at the output layer, and working its way to the input layer.

The error is distributed among the neurons based on their contribution to the output at each layer, and the weights and biases are adjusted to minimize the error. Gradient descent is used to perform this adjustment, which computes the gradient of the error with respect to the weights and biases and updates them in the direction of steepest descent (Wang Q. , 2020).

Compute the gradient of the error with respect to the weights and biases, the backpropagation algorithm employs the chain rule of calculus. This entails computing the derivative of the activation function used in each neuron, as well as the derivative of the error with respect to each neuron's output. The computations are conducted layer by layer, beginning with the output layer and progressing to the input layer (Wang Q. , 2020).

**Loss Function:** The loss function directs the optimization algorithm that adjusts the network's weights and biases during training. The optimization algorithm works by minimizing the value of the loss function, which is accomplished by adjusting the weights and biases in such a way that the difference between the predicted and true output is reduced. Backpropagation is the process of calculating the gradient of the loss function with respect to each weight and bias in the network. The optimization algorithm then adjusts the weights and biases in such a way that the network moves in the direction of the gradient, with the goal of lowering the loss function value (Wang Q. , 2020).

The SoftMax cross-entropy loss function is the most used loss function in speaker recognition. This loss function is used in neural network-based speaker recognition systems for speaker verification and identification (Author, 2023) (Wang Q. , 2020).

Formula:  $L(y_i, y)$

Where: L = Loss Function

$y_i$  = Predicted Output

Y = Actual Output

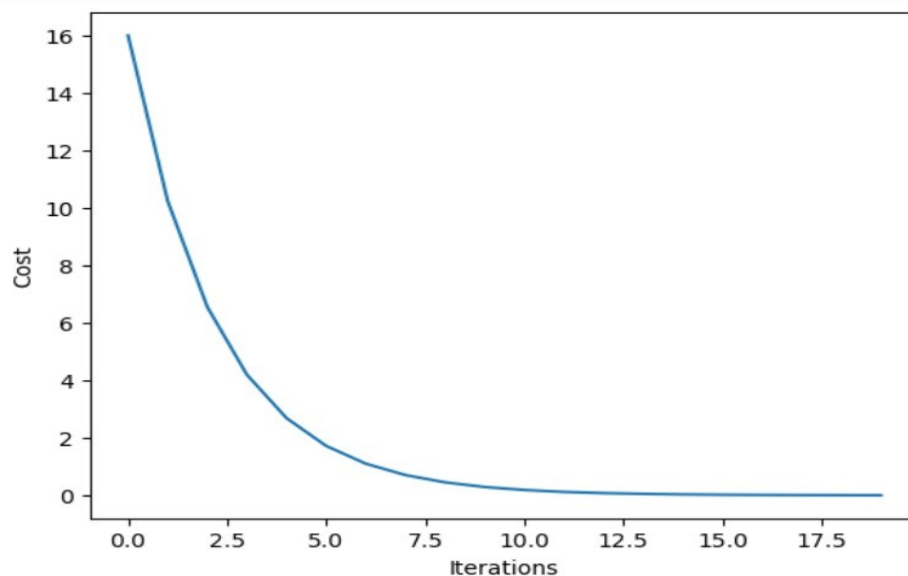
**Learning Rate:** The learning rate is a hyperparameter that determines how much to alter the model each time the model weights are updated in response to the estimated error.

It can be difficult to choose the learning rate because a value that is too small could lead to a lengthy training process that could become stuck, whereas a value that is too large could lead to learning a suboptimal set of weights too quickly or to an unstable training process (Brownlee, Understand the Impact of Learning Rate on Neural Network Performance, 2020).

**Gradient Descent:** The basic idea behind gradient descent is to iteratively update the network's parameters (weights and biases) in the direction of the cost function's negative gradient. The gradient is a vector of partial derivatives of the cost function with respect to each parameter that indicates the direction of the cost function's steepest increase (Brownlee, Understand the Impact of Learning Rate on Neural Network Performance, 2020).

Using the chain rule of calculus, the gradient of the cost function is computed with respect to each parameter during each iteration. The weights and biases are then updated in accordance with the following rule: Formula:  $w = w - \text{learning\_rate} * \text{gradient}$

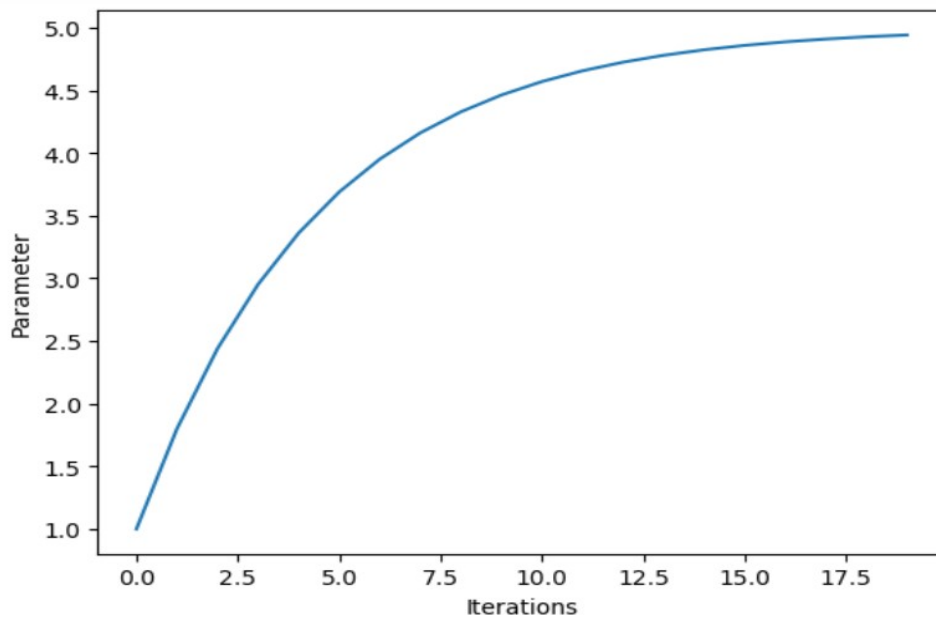
$b = b - \text{learning\_rate} * \text{gradient}$



**Figure 10. Plot for Gradient Descent.**

**Explanation:**

The value of the cost function is plotted over time as the gradient descent algorithm updates the parameter to minimize the cost. The x-axis represents the number of iterations, while the y-axis represents the cost function value. The cost function in the example code decreases rapidly at first, then levels off as the algorithm approaches the optimal value of the parameter.



**Figure 11. Plot for Parameters vs Iterations.**

**Explanation:**

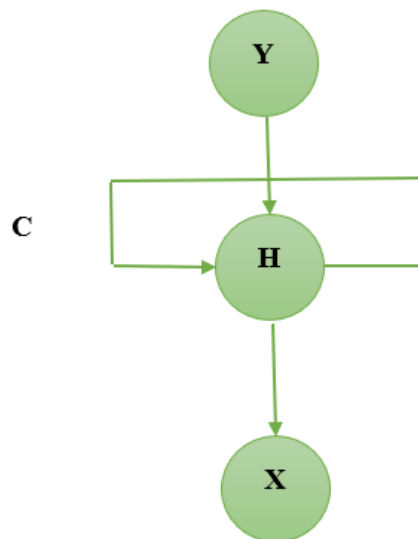
This graph depicts the evolution of the parameter (in this case,  $x$ ) as the gradient descent algorithm updates it to minimize the cost. The x-axis represents the number of iterations, while the y-axis represents the parameter value. We would like to see the parameter's value approach the optimal value that minimizes the cost function. We can see in the example code that the value of  $x$  increases rapidly at first and then levels off as it approaches the optimal value of 5.



## 6.1. Recurrent Neural Network:

Due to their capacity to simulate temporal dependencies in sequential data, recurrent neural networks (RNNs) are becoming increasingly common in speech processing applications. Speech recognition, speaker identification, speech synthesis, and speech emotion recognition are just a few of the speech processing tasks that RNNs can be used for (Le Zhang, 2017).

Many RNN architectures, including Long Short-Term Memory (LSTM) and Gated Recurrent Units, have been employed for speaker recognition. these architectures have demonstrated promising results, with some studies reporting high accuracy rates of up to 98%. (Chung, 2018). Speech signals can vary in length depending on the utterance, making it critical for speaker recognition algorithms to be able to manage variable-length input sequences. Enhance RNNs' performance on speaker recognition tasks, a variety of optimization techniques, such as gradient descent and backpropagation through time, can be used during the training process (Wang D. L., 2018).



**Figure 12. Basic RNN Representation.**

**Explanation:**

Above Network shows the input layer is "X," the hidden layer is "H," and the output layer is "Y" in this instance. The network parameters A, B, and C are used to enhance the model's output. "C" is the back propagation of the network.

**6.1.1. Parameter Efficiency:**

The advantage of RNNs is their ability to produce high accuracy with only a small amount of training data. This is so that they can learn complex patterns in the data that might not be obvious from a simple signal analysis because they can efficiently capture the temporal dependencies in speech signals (Wang Q. , 2020).

**6.1.2. Time Efficiency:**

RNNs can also be trained more quickly than traditional machine learning models on large datasets, which can be computationally intensive and time-consuming. This is because RNNs can effectively learn from small batches of data and make use of previously learned information to optimize the training process (Zafar, 2017).

**6.1.3. Robustness:**

RNNs are more resistant to changes in the speech signal like accent, speaking rate, and background noise. They can generalize to new examples of speech signals that may differ from those seen in the training data because they can effectively model the underlying structure of the speech signal (Zafar, 2017) (Fernández Gallardo, 2019).

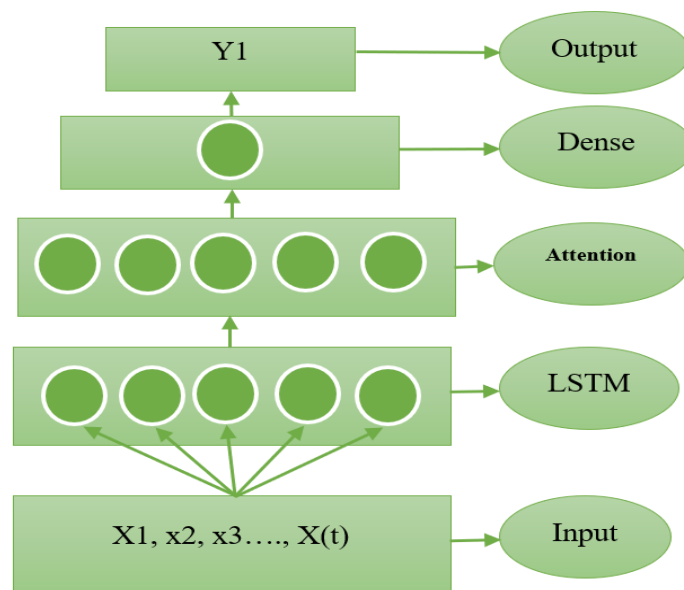
**6.1.4. Feature Extraction:**

Additionally, feature extraction, a crucial stage in speaker recognition can be accomplished using RNNs. An RNN can learn to extract features that are pertinent for speaker recognition tasks by being trained on a large corpus of speech data, which can increase the system's accuracy and robustness (Fernández Gallardo, 2019).

## 6.2. LSTM (Long Short-Term Memory):

Long Short-Term Memory, or LSTM, is a subset of RNNs (recurrent neural networks). It is built on a system of memory cells, each of which has a longer retention period. This device has Three gates—an input gate, an output gate, and a forget gate—control each memory cell. The amount of information to be input into the memory cell is determined by the input gate, the amount to be deleted from the memory cell by the forget gate.

The amount to be output from the memory cell is determined by the output gate (Wang D. L., 2018). For managing long-term dependencies in sequence data, the LSTM mechanism enables the network to selectively remember or forget the information from previous time steps. Since LSTM networks can gradually learn to recognize pertinent features in the audio data, they are particularly well-suited for speaker recognition (Biswal, 2023) (Wang D. L., 2018) (Wang Q. , 2020).



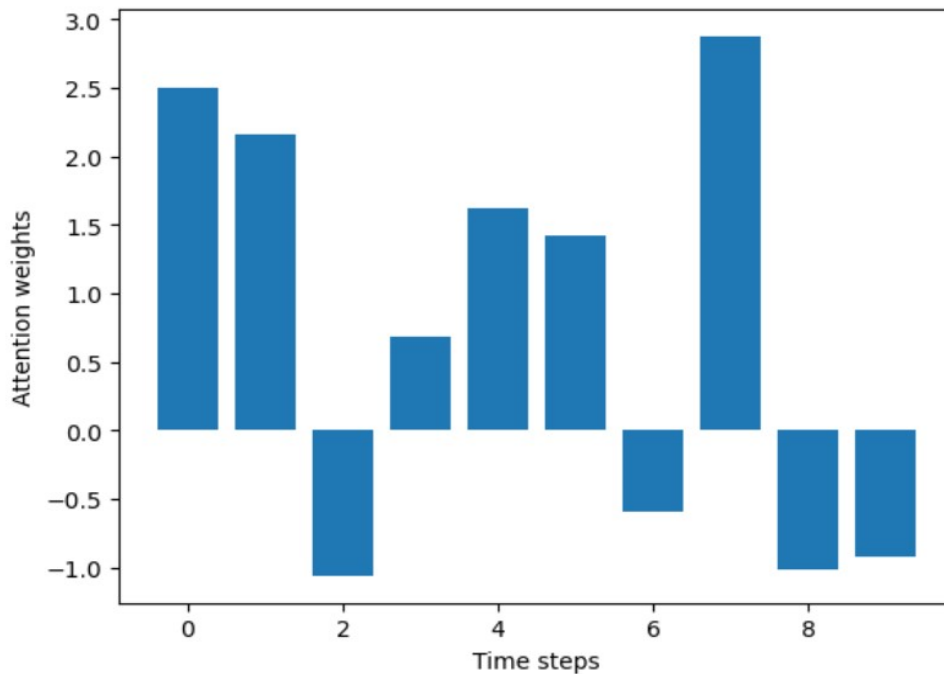
**Figure 13. LSTM Mechanism used in RNN.**

**Explanation:** The first layer is the input layer with  $x_1, x_2, x_3, \dots$ , which is followed by the LSTM hidden layer, the attention layer, the Dense layer, and the output layer, which is  $Y_1$ .

### 6.3. Attention Mechanism:

The attention mechanism is a method that enables the network to selectively focus on various elements of the input sequence while processing it. The tasks of speech recognition and natural language processing (NLP) have both made extensive use of this mechanism. The attention mechanism can also help speaker recognition, which is the process of finding which individual is speaking in an audio recording (Okabe, 2018).

In speaker recognition, an audio recording of a speech segment serves as the input, and the speaker identity is the model's output. The speaker's intonation and pronunciation are two aspects of the audio signal that the attention mechanism can help the model focus on to identify the speaker. The accuracy and resistance of the model to noise and changes in the speech signal may be enhanced as a result (Okabe, 2018). There are other Neural networks architectures where the Speaker verification and Speaker Identification can be done but to have more precise and effective neural network architecture RNN is the best choice for Speaker recognition with the combined LSTM and Attention Mechanism (Biswal, 2023) (Wang Q. , 2020).



**Figure 14. Attention weights of attention layer.**

**Explanation:**

The x-axis represents the input sequence's time steps, and the y-axis represents the attention weights assigned by the attention layer to each time step. Because the attention weights have been normalized to sum to one, the y-axis ranges from 0 to 1. Display the attention weights, we used a bar chart, with each bar representing the attention weight for a single time step. The height of each bar represents the attention weight value, and we can see that the attention weights are uniform across all time steps.

The goal of visualizing attention weights is to learn how the neural network processes input data when making predictions. Higher attention weights indicate that the model is emphasizing that time step, whereas lower attention weights indicate that the model is ignoring that time step. We can identify which parts of the input sequence are most relevant for making accurate predictions by analysing the attention weights and potentially improving the model's performance by adjusting the attention mechanism or incorporating additional features.

**7. Speaker Recognition System threats and Limitations:**

There are numerous dangers with all the background noise, other people speaking, and the intensity of the speaker's voice, it is impossible to identify the real speaker, not even with the most innovative technology like Deep Neural Networks. or even with the speaker's accent, dialects, pitch, and rhythm. Also, to consider a speaker's emotional state. This section of the paper discusses the theoretical aspects of potential system threats posed by attacks. A case study of these papers will demonstrate how to defeat these threats to a certain minimal level (Kamil A. Kamiński, 2022).

Speaker recognition systems may struggle to recognize speakers whose voices have changed due to aging, illness, or injury. Individuals' vocal cords may lose elasticity and become thinner as they age, resulting in changes in pitch and tone. Likewise, illness or injury to the vocal cords or respiratory system can cause changes in a person's voice (Wang Q. , 2020) .

## **7.1. Imposter Attacks:**

In impostor attacks, a person tries to impersonate another person's voice to gain unauthorized access, speaker recognition systems are susceptible. Direct attacks and indirect attacks are the two main categories of impostor attacks on speaker recognition systems. In a direct attack, the impostor tries to sound exactly like the target speaker to access their account or information. By altering the input signal, such as by adding noise or using a different microphone, an impostor attempts to control the speaker recognition system itself in an indirect attack (Gao, 2021).

### **7.1.1. Feature-level analysis:**

Anomaly detection works by analysing the extracted features of the speech signal, such as the frequency components, to identify any deviations or anomalies from the legitimate speaker's expected pattern. The system learns the expected pattern from the training data using machine learning algorithms, and then compares the features of the incoming speech signal to this pattern to determine whether it matches the legitimate speaker or not (Bengio, 2016).

If the incoming speech signal's features deviate significantly from the expected pattern, the system may flag it as an anomaly and initiate an alarm or additional verification steps to prevent unauthorized access. This can help to protect the system from impostor attacks while also increasing its overall security and reliability (Wang Q. , 2020).

### **7.1.2. Model-Based approaches:**

Based on differences in speech patterns between the target speaker and the impostor, these approaches use statistical models to identify impostor attacks (Gao, 2021) (Wang Q. , 2020). Statistical modelling works by comparing the speech patterns of the target speaker and the impostor. Machine learning algorithms are used by the system to learn statistical properties of the target speaker's voice from training data, such as the probability distribution of speech features (Gao, 2021).

When an impostor attempts to imitate the target speaker, their speech patterns will usually differ in some way, such as frequency components, phoneme duration, or other acoustic features. The system compares the speech patterns of the incoming speech signal to those of the target speaker and calculates the likelihood that the signal was produced by the target speaker using the learned statistical models (Gao, 2021).

### **7.1.3. Fusion-Based Methods:**

Multimodal approaches combine data from multiple sources, such as speech, speaker traits, and environmental factors, to improve the robustness and reliability of the speaker recognition system. To further verify the speaker's identity, the system may use other biometric modalities such as facial recognition or fingerprint scanning, as well as contextual information such as location, time of day, and device information, in addition to analysing the speech signal (Wang Q. , 2020).

Multimodal approaches can provide a more comprehensive and accurate representation of the speaker's identity by combining multiple sources of data, which can help to reduce the likelihood of false rejections or false acceptances. Furthermore, these approaches can aid in the detection of impostor attacks by identifying inconsistencies or discrepancies in data across multiple modalities.

### **7.1.4. Deep Neural Networks:**

Deep Learning techniques are used in these methods, which are based on deep learning, to identify patterns in speech signals and tell real speech from fake (Wang Q. , 2020).

## **7.2. Speaker Variability:**

The range of individual differences in speech characteristics is referred to as speaker variability. These variations can be found in the speech rate, pitch, tone, accent, and pronunciation. In a variety of disciplines, including speech recognition, speaker identification, and language acquisition, it is crucial to comprehend speaker variability (Gao, 2021) (Wang Q. , 2020).

Accent is a significant factor in speaker variation. Accent is the distinctive way a person pronounces words and phrases. It is influenced by things like the native language of the speaker, the local dialect, and personal speech patterns. Accent can significantly affect how speech is perceived and understood, especially in cross-cultural communication situations, according to research (Okabe, 2018) (Wang Q. , 2020).

Speech rate is another variable that affects speaker variation. This is a reference to the rate of speech. According to research, each person has a different preferred speaking rate, and these rates can change depending on the situation and the audience. Additionally, factors like emotional state, mental load, and speaking style can have an impact on speech rate (Wang Q. , 2020).

In many areas, including speech recognition and language acquisition, speaker variability is a crucial factor. For instance, to increase accuracy in speech recognition systems, it is crucial to consider individual variations in accent and pronunciation. Exposure to a variety of speakers with various accents and speaking tenors can aid language learners in improving their listening and comprehension abilities (Wang Q. , 2020).

### **7.3. Privacy Concern:**

While speaker recognition systems have several advantages, they also cause serious privacy issues. The main privacy issues with speaker recognition systems are listed below, along with references to pertinent studies.

#### **7.3.1. Biometric Data Collection:**

Biometric data, such as voiceprints, can be considered sensitive personal data because it can be used to uniquely identify an individual. If such information is collected without the individual's knowledge or consent, it may be misused or fall into the hands of the wrong people, resulting in identity theft or other negative consequences. As a result, it is critical to ensure that biometric information is collected, stored, and used in a transparent and ethical manner. Before collecting voiceprints, organizations that use speaker recognition software should obtain informed consent from individuals and be transparent about how the information will be used, stored, and protected (Shiva Prasad H. C., 2018).



### **7.3.2. Voice Data Storage:**

Voice data is sensitive personal information that must be safeguarded against unauthorized access, theft, or misuse. If someone gains access to this data, they could use it to impersonate the individual or access their personal information, resulting in identity theft or other negative consequences. Mitigate this risk, organizations that use speaker recognition systems should put in place strong security measures to protect the data. Data encryption, secure storage, and access controls are all part of this. Additionally, businesses should have policies and procedures in place to detect and respond to potential data breaches as soon as possible. (Singh, 2020).

### **7.3.3. Audio Surveillance:**

Governments and businesses must abide by applicable laws and regulations when using speaker recognition systems and recording audio. Many countries have laws that protect individuals' privacy and prohibit the unauthorized recording of conversations (Singh, 2020).

The government may have legal authority to use speaker recognition technology to record conversations in some cases, such as law enforcement investigations or national security matters, but this must be done within the bounds of the law and with appropriate oversight. Businesses must follow data protection laws and regulations, such as the European Union's General Data Protection Regulation (GDPR) or the California Consumer Privacy Act (CCPA) in the United States. These laws require businesses to be transparent about the collection and use of personal information, including voice data, and to obtain individuals' informed consent (Shiva Prasad H. C., 2018) (Singh, 2020).

### **7.3.4. False Positive identification:**

Reduce the risk of false positive identifications, use high-quality audio inputs, and perform regular system maintenance and calibration. Human oversight and review should also be included in the identification process to ensure that any potential errors are detected and corrected. Furthermore, people should be aware of the risks associated with speaker recognition systems and take appropriate precautions to protect their privacy (Wang Q. , 2020).

This includes keeping an eye on their accounts for any unusual activity, employing strong passwords, and reporting any inaccuracies or false positives to the appropriate authorities. It is also critical to have mechanisms in place to challenge false accusations and seek restitution for any harm caused (Wang Q. , 2020).

#### **7.3.5. Discrimination:**

Biases in speaker recognition systems can occur for a variety of reasons, including insufficient representation of diverse voices in the training data used to develop the system, differences in speech patterns across different demographics, or the use of algorithms that are not designed to account for differences in speech across different groups (Zhao, 2019).

Such biases can result in inaccurate or discriminatory identification of individuals, potentially causing harm and violating their rights. As a result of such biases, certain groups may be more likely to be misidentified or denied access to services or opportunities. Address these concerns, it is critical that diversity and inclusion considerations be incorporated into the development and deployment of speaker recognition systems. This can include ensuring that training data is representative of diverse voices, employing algorithms designed to account for differences in speech across diverse groups, and monitoring and assessing the system's accuracy and potential biases on a regular basis (Zhao, 2019).

Individuals who use speaker recognition systems should also be aware of these potential biases and take the necessary precautions to protect their privacy and rights. This includes checking their accounts for inaccuracies or discrimination and reporting any problems to the appropriate authorities (Zhao, 2019).

#### **7.4. Ethical considerations:**

Important ethical considerations for speaker recognition systems include transparency and explain ability. These systems ought to be created so that the user can understand how they function and how decisions are made (Doshi-Velez, 2017).

When these systems are used in risky applications like security and authentication, this is especially crucial. Users should be able to comprehend how the system determines their identity, and feedback should be given when mistakes are made (Doshi-Velez, 2017).

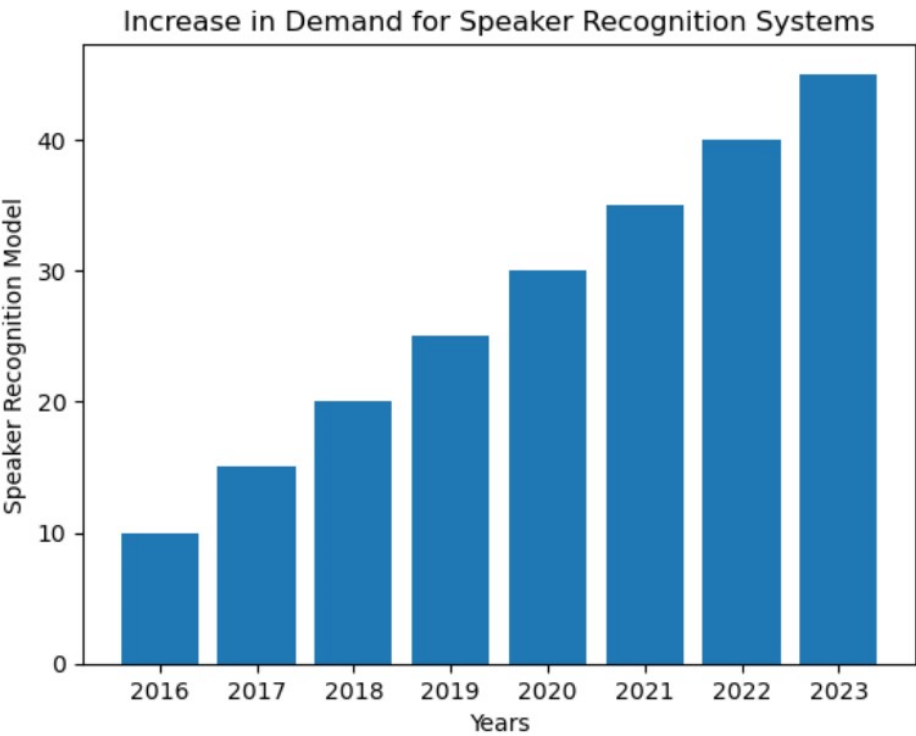
Promote fairness in speaker recognition systems, make sure the data used to develop them is diverse and representative of different voices, and that the algorithms used to analyse the data are designed to account for differences in speech across diverse groups. Identify and address any issues, the system's accuracy and potential biases must be monitored and evaluated on a regular basis (Oh, 2020).

## **8. Market Survey on Speaker recognition system:**

Deep learning models have gained popularity in speaker recognition systems in recent years due to their ability to learn complex patterns in speech signals. Convolutional neural networks (CNNs), recurrent neural networks (RNNs), and their variants such as long short-term memory (LSTM) networks and gated recurrent units (GRUs) have all been used in speaker recognition systems (Wang Q. , 2020). Recognition of Speech The market is expected to grow at a 15% CAGR from 2022 to 2028, surpassing USD 3.5 billion in 2021. Market leaders offer tailored solutions that boost corporate productivity by automating interactions between businesses and their customers.

The increasing use of voice biometrics in conjunction with real-time speech analytics in a variety of industrial verticals will drive market demand. Several companies are combining technologies with speech recognition systems to make it easier for businesses to implement solutions that augment their contact centres with an AI-based digital workforce. Five9, for example, announced in July 2021 that it will add real-time speech analytics, voice biometrics, and partner capabilities to its smart cloud contact centre system, allowing Five9 Intelligent Virtual Agents (IVAs) to be deployed with minimal scripting. It provides several pre-built IVA solutions for healthcare organizations that aid in health plan enrolment, prescription management, and appointment scheduling (GMI, 2022).

Researchers use evaluation metrics such as EER, minDCF, and DER to compare the performance of different deep learning models for speaker recognition. EER is the point at which the false acceptance and false rejection rates are equal, whereas minDCF is a metric that considers the costs of false acceptance and false rejection errors. DER is the metric used for speaker diarization or determining who spoke when in an audio recording (GMI, 2022).



**Figure 15. plot for rising demand in industry.**

**Explanation:**

The plot is a straightforward bar graph, with each bar representing the demand for speaker recognition systems in a specific year. The height of the bar represents the year's increase in demand, while the width of the bar remains constant. The plot's title, "Increase in Demand for Speaker Recognition Systems," clearly conveys the plot's purpose. The x-axis is labelled "Years," and the y-axis is labelled "Speaker Recognition Model Demand Increase," providing more information about the data represented by the plot. From 2016 to 2023, the plot shows a gradual increase in demand for speaker recognition systems. The demand for speaker recognition technology has steadily increased from 10 in 2016 to 45 in 2023, indicating a growing interest in the technology over time.

In the further Sections of market research concerning speaker recognition security issues and their demand will be explained this was just the basic explanation and plot to show the current demand and industry needs for the technology.

### **8.1. False acceptance and false rejection rate:**

False acceptance and rejection rates are important performance indicators for speaker recognition systems. When an unauthorized person is granted access, false acceptance occurs, and false rejection occurs when an authorized person is denied access. According to a survey conducted by the National Institute of Standards and Technology (NIST), false acceptance rates are a major concern for both consumers and businesses, and lowering false acceptance rates is a key driver for improving speaker recognition technology (Sullivan, 2017). The need for more secure authentication methods, as well as the growing adoption of voice-activated technologies such as virtual assistants, are driving the rising demand. Furthermore, advances in deep learning and artificial intelligence have increased the accuracy and efficiency of speech recognition systems. Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Deep Belief Networks (DBNs) have shown promising results in speaker recognition, particularly in noisy environments (Sullivan, 2017).

### **8.2. Compliance with regulations:**

For businesses and organizations deploying speaker recognition systems, regulatory compliance is critical. Compliance with regulations such as GDPR and CCPA is critical to ensuring user data privacy and security. According to a Frost & Sullivan survey, regulatory compliance is a key driver for speaker recognition technology investment, and organizations are willing to pay a premium for solutions that meet regulatory requirements. The GDPR is a European Union regulation that establishes stringent requirements for data protection and privacy. It applies to any organization that processes the personal data of EU citizens. The CCPA is a California law that requires businesses to give consumers more control over their personal data (Sullivan, 2017) (GMI, 2022).

## 9. Datasets:

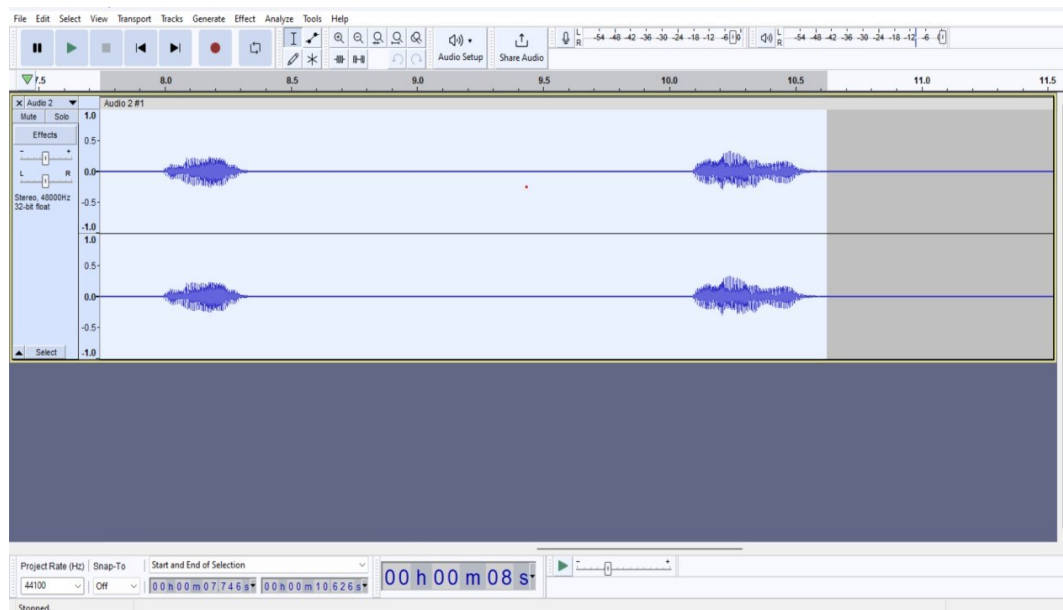
VoxCeleb1, VoxCeleb2, Common Voices, TIMIT etc., provide many varieties of datasets ranging from 10 speakers with 100 utterances to 100,000 speakers with 1 million utterances. Latest advancement shows that the 176 billion parameters were trained for the language for many such requirements and specific tasks like precise machine translation, named recognition entity, emotional analysis etc. (OpenAI, 2020). Datasets without proper training features and labels can be unstructured to pattern viewed by model.

The Dataset which I propose in this paper is a custom dataset with six speakers and 420 utterances (Multiple Languages). Results have shown that by Using a custom Dataset to train GMM, SVM and RMM models can be more efficient and have better results. Large datasets may also incur computational costs because processing and training the model on them demands more resources. Additionally, the system's performance may be impacted by the inherent biases present in some datasets, such as an imbalance in the number of speakers or differences in the sound quality of the recordings (Bengio, 2016).

The dataset consists of audio recordings of different speakers recorded in a closed environment with 0.5 meters, 1 meter, 1.5 meters, 2 meters, 2.5 meters and with minimal background noise. The main idea is to recognize the speaker's voice even from 1 – 2.5 meters in different environments. As Spoken in the previous sections about the speaker recognition system security threats for having to reduce the imposter attacks (unknown speaker) and speaker variability. Both limitations can be reduced, and the efficiency of the system can be increased by a dataset which has multiple utterances with multiple languages. The benefits of these are explained in further sections.

In the next Sections we will see about the implementation of the code example of converting a raw audio signal to the format where the models take as input and then make the predictions, compare the result with the model trained with single language of all speakers to multiple language of all speakers.

## 9.1. Recording Audio Signals:



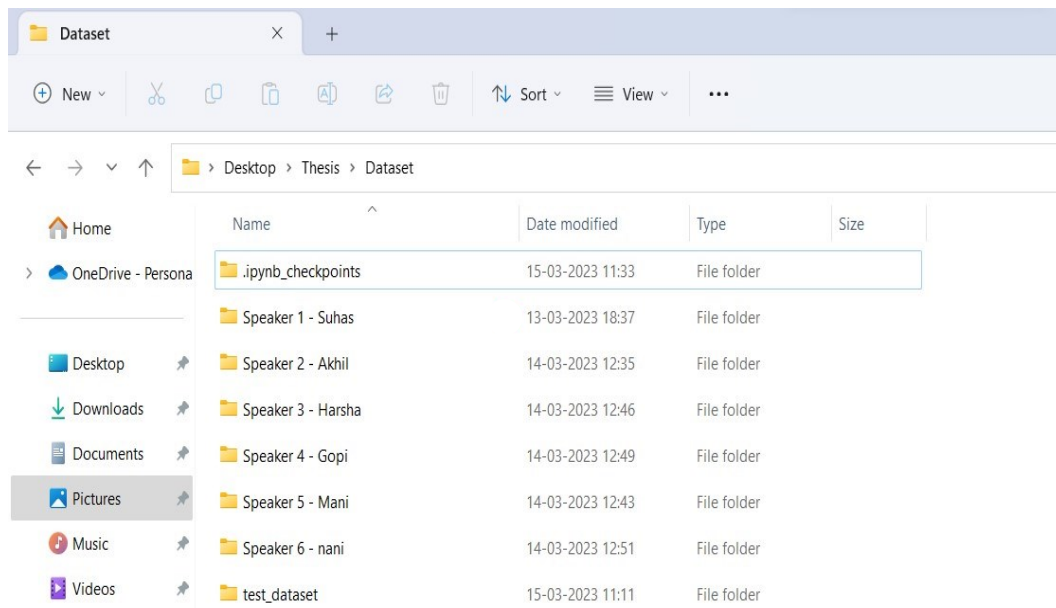
**Figure 16. Audacity Software for Recording Audio Samples.**

### **Explanation:**

Audacity is a free and open-source digital audio editor and recorder. It is available for Windows, Mac, and Linux computers. Audacity can be used to record and mix live audio, as well as edit and manipulate digital audio files. Audio Recordings are being stored into the device with .WAV format and with sampling frequency of 44100Hz. Audacity is a Software where the waveform can be generated at any pitch desired and for further analysis the Spectrogram of the can also be visualized.

Some examples of common applications are Podcasts and other audio content are recorded and edited. Digitizing and restoring old tapes or recordings, Making, and editing music tracks, as well as mixing and mastering., Producing sound effects and other audio elements for use in videos or multimedia projects, getting rid of unwanted noise or background sounds in recordings, Converting between various audio formats or file types, Performing audio signal analysis or measurement.

## 9.2. Creating a Directory:



**Figure 17. Directory creating and Storing the Audio recordings.**

### Explanation:

The Directory: '\\Users\\Suhas\\Desktop\\Thesis\\Dataset' is storing the respective Speaker's Audio recordings by their names with a folder, each audio recording is labelled "language\_ (distance in meters) \_BGN (background noise) \_ (Name of the speaker).wav. For example, "**spanish\_1D0BGNSUHAS.wav**" would be the speaker - Suhas speaking Spanish to the system from one meter with no background noise. Create a dataset of Unknown and known speaker for and store them for further model building and testing of the model accuracy and loss function.

The feature extraction which is Mel-Frequency Cepstral Coefficients can be done on the recorded dataset and then concatenation of the feature extracted files can be saved to different directory. After the proper feature extraction, the files are labelled as '**(Speaker Name) \_(language). Npy**' is the fully feature extracted format, which is MFCC, and we will see that in the coming sections.



### 9.3. Benefits of Multi-lingual Datasets:

A multi-language dataset can aid in enhancing the speaker recognition system's resistance to linguistic variations. The model can learn to recognize common speech patterns between languages and become better at differentiating between speakers, even if they speak different languages, by being trained on speech from a variety of languages (Martinez, 2014). When used with languages for which there is little training data, the model can learn to recognize universal speech patterns, which can help to increase the system's accuracy. which can lessen bias and enhance the speaker recognition system's generalization performance (Wang Q. , 2020).

**Table 1: First Data with Zero Background Noise Recordings.**

Languages	Utterances
English	Hello, how are you?
German	Hallo, Wei Geht's?
French	Bonjour, comment allez-vous
Spanish	Hola, Como estas
Russian	Привет как дела
Kannada	ಹಲೋ, ಹೋಗಿದ್ದೀಯ
Telugu	హలో, ఎలా ఉన్నారు
Tamil	வணக்கம் எப்படி இருக்கிறாய்
Hindi	नमस्ते, आप कैसे हैं
Malayalam	നിങ്ങളിക്ക് സുഖമാണ ൾ

**Explanation:**

The above table shows that the different languages have different accents, dialects, pitch, and tone, which in turn helps in reducing the variability effect on the speaker recognition system and improved efficiency for security from imposter attacks. Dataset Consisting of 10 Languages in which 3 are European (German, French and Spanish), and 5 of the Indian Languages (Kannada, Telugu, Tamil, Malayalam, and Hindi) and English and Russian.

**9.3.1. Data with 0.5 meters distance:****Table 2: Easy Utterance dataset with minimal distance.**

Languages	Utterances
English	I am Good
German	Mir geht es gut
French	je vais bien
Spanish	Soy bueno
Russian	У меня все хорошо
Kannada	ನಾನು ಚೆನಾಗಿದೆದೇನೆ
Telugu	నేను భాగుననాను
Tamil	நான் நன்றாக இருக்கிறேன்
Hindi	टीक हनु
Malayalam	എനിക്ക് സുഖമാണ്

**Explanation:**

This dataset was created using the measuring tape from the recording device and the speaker, regardless of whether the speaker was speaking in normal tone of the distance was just to capture the intensity of the speech signal and to train the model.

### Explanation:

Based on this signal where the different model architectures are trained for on the testing set of data and gives an answer based on that. There was no background noise in these recordings because they were made in a closed environment, regardless of how little background noise was created outside the room or environment. Because there are ten different languages, it was difficult for the speaker to keep up with the pronunciation of the words. The main idea is to capture the tone, accent, and dialects that change automatically for different languages when spoken to keep the recordings between 3 - 4 seconds.

### 9.3.2. Data with 1 meter distance:

**Table 3: Distance of 1 meter between speaker and recording device.**

<b>Languages</b>	<b>Utterances</b>
<b>English</b>	I am Sorry
<b>German</b>	Es tut mir leid
<b>French</b>	je suis désolé
<b>Spanish</b>	lo siento
<b>Russian</b>	мне жалъ
<b>Kannada</b>	ನನಾನು ಕ್ಷಮಿಸು
<b>Telugu</b>	నను క్షమించిండి
<b>Tamil</b>	என்னன மன்னிக்கவும்
<b>Hindi</b>	माफी चाहता
<b>Malayalam</b>	എന്നാട് ക്ഷമിക്കൂ

**Explanation:**

When audio recordings are taken from one meter away from a device and speaker with no background noise, the resulting recordings can be of high quality and suitable for speaker recognition tasks. Because the audio signal has more room to propagate before reaching the microphone, recording from one meter can help to minimize distortion and reverberation effects. Furthermore, no background noise can help to improve the recording's signal-to-noise ratio, making it easier to distinguish between different speakers based on their distinct vocal characteristics. Ensure reliable speaker recognition results, ensure that the recording conditions are as consistent and controlled as possible (Wang Q. , 2020).

**9.3.3. Audio Recordings with 1.5-meter distance:****Table 4: Distance of 1.5 meters with Zero Background Noise.**

Languages	Utterances
English	Do not cry
German	weine nicht
French	ne pleure pas
Spanish	no llores
Russian	не плачь
Kannada	ಅಳಬೇಡ
Telugu	ఏడవకండి
Tamil	அழாறே
Hindi	रोओ मत
Malayalam	കരയരുത്

**Explanation:**

Recording from 1.5 meters away can still help to minimize distortion and reverberation effects, but it may result in a slightly lower signal-to-noise ratio than recordings from a closer distance. This can be mitigated by using a higher-quality microphone or increasing the recording volume. The recording's intensity (measured in decibels or dB) will be affected by several factors, including the loudness of the speaker's voice, the distance between the speaker and the microphone, and the sensitivity of the microphone itself.

The intensity of the recording may be slightly lower at 1.5 meters than at closer distances, but it can still be kept at a normal level by adjusting the microphone gain or using a microphone with a higher sensitivity. The recording's energy (measured in joules or J) will be affected by the same factors as the intensity, but it may be less affected by distance because it is related to the total amount of sound energy present in the recording (Marcos Faundez-Zanuy, 2022).

**9.3.4. Audio Recordings with 2 meters distance:****Table 5: 2 Meters distance to speech signal recordings and system.**

<b>Languages</b>	<b>Utterances</b>
<b>English</b>	I am Hungry
<b>German</b>	ich bin hungrig
<b>French</b>	j'ai faim
<b>Spanish</b>	tengo hambre
<b>Russian</b>	я голоден
<b>Kannada</b>	ನನಗೆ ಹಸಿವಾಗಿದೆ
<b>Telugu</b>	ನನకు ఆకలిగా ఉంది
<b>Tamil</b>	எனக்கு பசிக்கிறது
<b>Hindi</b>	मझे भु ख लगी है
<b>Malayalam</b>	എനിക്ക് വിശക്കുന്നു

**Explanation:**

In terms of speaker recognition, using audio recordings with no background noise can help improve algorithm accuracy. This is because background noise can interfere with the audio signal and make identifying specific vocal characteristics that are unique to a particular speaker more difficult. It is critical to capture the audio recording at a sufficient volume and maintain a consistent distance between the speaker and the recording device throughout the recording to improve the accuracy of speaker recognition.

Additionally, signal processing techniques may be useful for filtering out any unwanted noise or distortion in the audio signal. However, if no obstructions or interference exist, the intensity and energy of the sound should remain normal (ABU QUWSAR OHI, 2021).

**9.3.5. Audio Recordings with 2.5 meters distance:****Table 6: 2.5 Meters Distance to speech signal recordings and system.**

Languages	Utterances
English	You are beautiful
German	du bist schön
French	tu es belle
Spanish	Eres hermoso
Russian	ты красивый
Kannada	ನೇನು ಸುಂದರವಾಗಿ ಇರುವೆ
Telugu	నువ్వు అందంగా ఉన్నా వ్వ
Tamil	நீ அழகாக இருக்கிறாய்
Hindi	आप स ुंदर हैं
Malayalam	നിങ്ങൾ സുന്ദരിയാണ്

**Explanation:**

When an audio recording is made from 2.5 meters away from a device and speaker with no background noise, the sound waves must travel through the air and may encounter obstacles such as walls or other objects, causing the sound to become weaker or distorted. However, if no obstructions or interference exist, the intensity and energy of the sound should remain normal. The sound waves will have to travel a greater distance than if the recording was made from 2 meters, resulting in weaker sound waves at the point of recording.

This decrease in intensity will result in a lower signal-to-noise ratio, making it more difficult to identify specific vocal characteristics that are unique to a specific speaker. In terms of speaker recognition, using audio recordings with no background noise can help improve algorithm accuracy. This is because background noise can interfere with the audio signal and make identifying specific vocal characteristics that are unique to a particular speaker more difficult. It is critical to capture the audio recording at a sufficient volume and maintain a consistent distance between the speaker and the recording device recognition (Chung, 2018).

In speaker recognition systems, ensuring high-quality audio recordings is critical for accurate speaker identification. Reduced background noise and consistent recording conditions can significantly improve system performance and reduce the risk of misidentification or false positives. When capturing audio data for speaker recognition applications, it is also critical to consider the potential impact of environmental factors such as distance and obstructions (Gunawan, 2018).

### 9.3.6. Audio Recordings with minimal background noise:

Table 7: 0.5 Distance from device to speaker with background noise.

Languages	Utterances
English	I had eggs for breakfast
German	Ich hatte Eier zum Frühstück
French	j'ai eu des oeufs pour le petit déjeuner
Spanish	desayuné huevos
Russian	у меня были яйца на завтрак
Kannada	ನಾನು ಉಪಹಾರಕ್ಕಾಗಿ ಮೊಟ್ಟೆಗಳನ್ನು ಹೆಂಡಿದೆದ
Telugu	నేను అల్పా హారం కోసం గుడ్లు తీసుకున్నాను
Tamil	காளை உணவாக முட்டை சாப்பிட்டுடன்
Hindi	मेरे पास नाश्ते के लिए अंडे थे
Malayalam	എനിക്ക് പ്രഭാതഭക്ഷണത്തിന് മുട്ട ഉണ്ടായിരുന്നു

#### Explanation:

Sound waves will be strong and intense at the point of recording when an audio recording is made from 0.5 meters from a device and speaker with minimal background noise. This is because the sound waves will have had less time and distance to travel, resulting in less attenuation or reduction in intensity. Assuming there is little background noise, the intensity and energy of the sound should remain normal. However, if the recording device is not capable of handling prominent levels of sound pressure, the strong and intense sound waves may cause distortion or clipping.



## 10. Code Examples:

### 10.1. Feature Extraction (MFCC):

```
In [8]: import librosa

# load audio file
audio_file_path = 'german.wav'
y, sr = librosa.load(audio_file_path, sr=None)

# extract MFCC features
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)

print(mfccs.shape)
```

(13, 262)

**Code Example 1. Import Dependencies and Feature Extraction.**

#### **Explanation:**

In this code, we first use the `librosa.load` function to load an audio file. The audio signal is stored in the `y` variable, and the sampling rate of the audio file is represented by `sr`. The MFCC features are then extracted using the `librosa.feature.mfcc` function. The audio signal and sampling rate are represented by the `y` and `sr` parameters, respectively. In this example, the `n_mfcc` parameter specifies the number of MFCC coefficients to extract, which is set to 13. Finally, we print the `mfccs` array's shape, which is a 2D array of shape `(n_mfcc, n_frames)`. Each array column represents the MFCC coefficients for one frame of an audio signal. Same procedure to all the audio files individually or to mention the entire directory containing of all the audio files and feature extracting them.

The first dimension of `(13, 262)` represents the number of extracted MFCCs, which in this case is 13. The second dimension is the number of frames in the audio signal, which in this case is 262. Overall, the feature extraction can be done in a simple and effective manner later sections tells about padding all the audio signals into one length (Amatriain, 2023) (Kuldip kaur, 2015).

## 10.2. Padding all the feature extracted to same length:

```
In [6]: #Maximum Length of the padded feature vectors
max_length = 100

#iterate through each loop in the audio file's MFCC
padded_mfccs = []
for mfccs in mfcc_features:
    n_frames = mfccs.shape[1]
    if n_frames < max_length:
        padded_mfccs.append(np.pad(mfccs, (0, 0),
                                   (0, max_length - n_frames)), mode='constant')
    else:
        padded_mfccs.append(mfcc[: , :max_length])

padded_mfccs = np.stack(padded_mfccs, axis=0)
print(padded_mfccs.shape)
```

**Code Example 2. Padding all the feature Extracted files to fixed length.**

### Explanation:

We consider in this code that mfcc\_features are a list of 2D arrays, each of which contains MFCC features for a single audio file. Each MFCC array should be padded to a fixed length of max\_length. Accomplish this, we iterate through the MFCCs of each audio file and count the number of frames in the array (n\_frames). If n\_frames is less than max\_length, we use the numpy.pad function to pad the MFCCs with zeros along the second dimension. We truncate the MFCCs to max\_length frames if n\_frames are greater than or equal to max\_length. Finally, we use numpy.stack to convert the list of padded MFCCs to a 3D NumPy array, with the first dimension representing the audio file, the second dimension representing the MFCC coefficient, and the third dimension representing the frame (Wang Q. , 2020).

In the directory the data will be stored in a separate file with the same length since it is already feature extracted the data will be of .npz format and there are total of 420 .npz files of 6 speakers with 10 different languages (which is 7 utterances per language). In this case the size of the padded file is 216KB for all the files and the files are named as 'SpeakerName\_LanguageandDistance.npz' .

### 10.3. Concatenation of all language files with respective speaker:

```
In [15]: audio1 = np.load('C:/Users/Suhas/Desktop/feature_extraction/Akhil/English Akhil.npy')
audio2 = np.load('C:/Users/Suhas/Desktop/feature_extraction/Akhil/french Akhil.npy')
audio3 = np.load('C:/Users/Suhas/Desktop/feature_extraction/Akhil/German Akhil.npy')
audio4 = np.load('C:/Users/Suhas/Desktop/feature_extraction/Akhil/hindi akhil.npy')
audio5 = np.load('C:/Users/Suhas/Desktop/feature_extraction/Akhil/kannada akhil.npy')
audio6 = np.load('C:/Users/Suhas/Desktop/feature_extraction/Akhil/tamil akhil.npy')
audio7 = np.load('C:/Users/Suhas/Desktop/feature_extraction/Akhil/Telugu akhil.npy')
audio8 = np.load('C:/Users/Suhas/Desktop/feature_extraction/Akhil/spanish akhil.npy')
audio9 = np.load('C:/Users/Suhas/Desktop/feature_extraction/Akhil/Malayalam akhil.npy')
audio10 = np.load('C:/Users/Suhas/Desktop/feature_extraction/Akhil/Russian akhil.npy')







features = np.concatenate((audio1, audio2, audio3, audio4, audio5, audio6,
                           audio7, audio7, audio8, audio9, audio10), axis=0)

np.save('concatenated_features_Akhil.npy', features)
```

**Code Example 3. Concatenation of Feature extracted files.**

#### Explanation:

The feature extracted data from different languages of the same Speaker are stored in a directory and concatenated into NumPy array which will be easy for converging into machine learning model. Save the file to the same or different directory in this case **concatenated\_features\_Suhas.npy**. Same Process for each Speaker until the raw audio data is concatenated and normalized. In these sections the files with the single language are concatenated first ('SUHAS\_ENGLISH0.5D0BGN.npy') which means the speaker's name and language and the distance to device and speaker with or without the presence of background noise.

Name	Date modified	Type
 concatenated_features_Akhil.npy	24-03-2023 10:42	NPY File
 concatenated_features_Gopi.npy	24-03-2023 10:42	NPY File
 concatenated_features_Harsha.npy	24-03-2023 10:42	NPY File
 concatenated_features_Mani.npy	24-03-2023 10:42	NPY File
 concatenated_features_Nani.npy	24-03-2023 10:42	NPY File
 concatenated_features_Suhas.npy	24-03-2023 10:42	NPY File

**Figure 18. Proof of files stored in NPY format.**

## 10.4. Data Augmentation Technique:

Data augmentation is a machine learning and computer vision technique that uses various transformations on existing data samples to artificially increase the size and diversity of a training dataset. These transformations keep the underlying data while introducing variations like translations, rotations, scaling, and other changes. When the original dataset is limited or imbalanced, data augmentation can help prevent overfitting and improve the generalization capability of the trained models (Cubuk, 2019).

There are different types of data augmentation techniques, especially when we have a small dataset that was prepared with the requirements in mind (recordings with varying lengths, background noises, and so on). We add three techniques in data augmentation to further enrich the dataset with meaningful insights:

### 10.4.1. Speed Perturbation:

Speed perturbation, also known as tempo perturbation or time stretching, is the process of changing the speed of an audio signal by increasing or decreasing its duration while maintaining its pitch. This technique is commonly used to supplement training data and improve robustness against variations in speaking rate in speech recognition and speaker verification tasks (Ko, 2015).

```
In [7]: import numpy as np
import librosa

#apply new speed perturbations to features
def speed_perturbation(features, factor):
    new_features=[]
    for i in range(features.shape[0]):
        new_features.append(librosa.effects.time_stretch(features[i],factor))
    return np.array(new_features)

#iterate over each audio
for speaker_id in range(1, 7):
    features = np.load('AKHIL_ENGLISH.npy')

    augmented_features = speed_perturbation(features, 0.9)

    np.save(f'AKHIL_ENGLISH_AUGMENTED.npy', augmented_features)
```

**Code Example 4. Speed Perturbation for audio signals.**

### Explanation:

The speed perturbation (features, factor) function is defined. This function accepts a 2D array of features as well as a factor value. It uses the librosa.effects.time\_stretch function to apply speed perturbation to each feature and returns a new array of perturbed features. The code then iterates through each speaker using the numbers 1 through 7. This implies that six speakers (1, 2, 3, 4, 5, 6) are being processed. The features are loaded from a.npy file inside the loop using the np.load function.

'AKHIL\_ENGLISH.npy' appears to be the file name. After that, the loaded features are passed to the speed perturbation function with a factor of 0.9. This means that the features will be stretched by 10%, which will make them slower (Md Sahidullah, 2021).

#### 10.4.2. background Noise:

The addition of several types of environmental noise to clean audio signals is known as background noise augmentation. The goal is to make trained models more resistant to real-world conditions with background noise. Background noises that are commonly encountered include street noise, babble noise, and white noise. The audio recording has 60-75db with ambient noise effect or moderate noise environment (Virtanen, 2018).

```
In [8]: import numpy as np
import librosa

features = np.load('AKHIL.npy')

noise, sr = librosa.load('backgroundnoiseupdated.wav', sr=None)

#random selecting of the noise segment
start_idx = np.random.randint(len(noise) - len(features))
noise_segment = noise[start_idx : start_idx + len(features)]

desired_noise_level = 0.3
max_amplitude = np.max(features)
noise_segment = noise_segment * (desired_noise_level * max_amplitude) / np.max(noise_segment)

#Mix the noise segment with the features
augmented_features = features + noise_segment[:, np.newaxis]

np.save('AKHIL_BG.npy', augmented_features)
```

**Code Example 5. Addition of background noise to the speaker's voice.**

## Explanation:

The code chooses a segment of background noise at random that is the same length as the features. It generates a random start index within the valid range of the noise audio using `np.random.randint()` and then slices the noise array accordingly. The noise segment's volume is adjusted to a desired noise level (`desired_noise_level`). The code computes the maximum amplitude of the features (`max_amplitude`) and scales the noise segment relative to that maximum amplitude to match the desired noise level. By element-wise addition, the noise segment is combined with the features. The `noise_segment[: np.newaxis]` function resizes the noise segment to match the dimensions of the features array, allowing proper addition (musikalkemist, 2022).

### 10.4.3. Room Simulation:

Room simulation, also known as reverberation simulation, is the process of adding artificial room impulse responses (RIRs) to audio signals to simulate the effect of sound reflecting off room surfaces. This technique is important in tasks like speech recognition, source separation, and speech dereverberation because it trains models to deal with reverberant environments (Koutini, 2019).

```
In [10]: import numpy as np

#function for shifting the original data to room dimensions
def simulate_room(npy_file, room_dimesion, sound_speed):
    original_data = np.load(npy_file)
    original_shape = original_data.shape
    time_delay = np.sqrt(np.sum(np.square(room_dimensions)))/sound_speed
    samples_to_shift = np.zeros(original_shape)
    shifted_data = np.zeros(original_data)

    if samples_to_shift > 0:
        shifted_data[samples_to_shift:] = original_data[:-samples_to_shift]
    elif samples_to_shift < 0:
        shifted_data[:samples_to_shift] = original_data[-samples_to_shift:]
    return shifted_data

np_file = 'HARSHA.npy'
room_dimensions = [5,4,3]
sound_speed = 343

augmented_data = simulate_room(np_file, room_dimensions, sound_speed)
```

**Code Example 6. Room simulation code.**

## Explanation:

It is defined as the function `simulate room`, Three parameters are required for this function `numpy` file, `room_dimensions`, `sound_speed`. The location of the original. The audio data is stored in a `numpy` file. `room_dimensions`: A list or array of the room's dimensions in meters, given as `[length, width, height]`. `sound_speed`: Sound speed in meters per second. The shape of the original data is retrieved and stored in the `original_shape` variable using `original_data.shape`. The time delay is determined by the room dimensions and the sound speed.

The distance is divided by the sound speed using the following formula:  $\text{np.sqrt}(\text{np.sum}(\text{np.square}(\text{room\_dimensions}))) / \text{sound\_speed}$ . The number of samples to shift is determined by the time delay. `Int(time_delay * original_shape [0])` is obtained by multiplying the time delay by the number of samples in the original data. Using `np.zeros(original_shape)`, an empty array, `shifted_data`, is created with the same shape as the original data. Using conditional statements, the data is shifted by the calculated number of samples. If `samples_to_shift > 0`, the data is shifted to the right by slicing the arrays. If the value of `samples_to_shift` is less than zero, the data is shifted to the left. Finally, the function returns the shifted data. An example usage is provided at the end, where the `numpy_file` variable must be replaced with the path to your own. Set the `room_dimensions` to the dimensions of your room and specify the `sound_speed` in meters per second in the `numpy` file (David Diaz-Guerra, 2018).

## 10.5. Addition of All the Speakers Files with data augmented files:

```
In [14]: import numpy as np

file_names=['AKHIL_ENGLISH_AUGMENTED.npy', 'AKHIL_FRENCH_AUGMENTED.npy', 'AKHIL_GERMAN_AUGMENTED.npy',
            'AKHIL_HINDI_AUGMENTED.npy', 'AKHIL_KANNADA_AUGMENTED.npy', 'AKHIL_MALAYALAM_AUGMENTED.npy',
            'AKHIL_SPANISH_AUGMENTED.npy', 'AKHIL_RUSSIAN_AUGMENTED.npy', 'AKHIL_TELUGU_AUGMENTED.npy',
            'AKHIL_TAMIL_AUGMENTED.npy']

arrays =[]

#Load each file and append it's content
for file_name in file_names:
    data = np.load(file_name)
    arrays.append(data)

#concatenate the arrays along the first axis
combined_arrays = np.concatenate(arrays, axis=0)

np.save('AKHIL_AUGMENTED.npy', combined_arrays)
```

**Code Example 7. Concatenate all the Speaker's files.**

### Explanation:

Concatenating the Concatenated feature extracted files of different language of Speaker's to all the data augmented files into a single file by name AKHIL\_AUGMENTED.npy for one speaker and same for all. features.append adds all the files from the directory to a single file and saves them in feature extracted format to the specified location.

### 10.6. Combination of all the files of speakers:

```
In [15]: import numpy as np

file_names=['AKHIL_AUGMENTED.npy', 'GOPI_AUGMENTED.npy', 'HARSHA_AUGMENTED.npy',
            'MANI_AUGMENTED.npy', 'NANI_AUGMENTED.npy', 'SUHAS_AUGMENTED.npy']

arrays =[]

#Load each file and append it's content
for file_name in file_names:
    data = np.load(file_name)
    arrays.append(data)

#concatenate the arrays along the first axis
combined_arrays = np.concatenate(arrays, axis=0)

np.save('COMPLETE_DATA.npy', combined_arrays)
```

**Code Example 8. One complete dataset of all speakers.**

### Explanation:

The code specifies a list of file names (file\_names) that correspond to the.npy files that should be loaded and combined. To store the loaded arrays, an empty list, arrays, is created. The code then enters a loop that iterates through the file\_names list, one by one. The np.load(file\_name) function is used within the loop to load each.npy file, and the loaded data is stored in the data variable. The loaded array (data) is appended to the list of arrays. Following the completion of the loop, the arrays are concatenated along the first axis using the np.concatenate(arrays, axis=0) function. This assumes that the arrays are identical except for the first axis, and that concatenation is done vertically. The combined array that results is saved in the combined\_array variable. Finally, the combined\_array is saved to a new.npy file called "COMPLETE\_DATA.npy" by calling the np.save() function .



## 10.7. Train, Validation and Test, Features and Labels:

```
In [17]: import numpy as np
from sklearn.model_selection import train_test_split

combined_array = np.load('COMPLETE_DATA.npy')

#ratio split
train_ratio = 0.7
val_ratio = 0.15
test_ratio = 0.15

X_train, X_remaining = train_test_split(combined_array, train_size=train_ratio, random_state=42)

relative_ratio = val_ratio/(val_ratio+test_ratio)
X_val, X_test = train_test_split(X_remaining, train_size=relative_ratio, random_state=42)

print('Training set shape:', X_train.shape)
print('Validation set shape:', X_val.shape)
print('Test set shape:', X_test.shape)

np.save('TRAIN_UPDATED.npy', X_train)
np.save('VALIDATION_UPDATED.npy', X_val)
np.save('TEST_UPDATED.npy', X_test)

Training set shape: (112896, 479)
Validation set shape: (24192, 479)
Test set shape: (24192, 479)
```

### Code Example 9. Train, validation, and test sets.

#### Explanation:

`np.load()` is used to load the combined array from the "COMPLETE\_DATA.npy" file and store it in the `combined_array` variable. The data is divided into three sets: training, validation, and test. `Train_ratio` for the training set, `val_ratio` for the validation set, and `test_ratio` for the test set is all specified. Using `train_test_split()`, the data is first divided into a training set and the remaining data. Determine the data ratio used for the training set, the `train_size` parameter is set to `train_ratio`. The remaining data is saved in the variable `X_remaining`. Using `train_test_split()`, the remaining data is further divided into validation and test sets. Determine the data ratio used for the validation set, the `train_size` parameter is set to `relative_ratio`, which is calculated as  $\text{val\_ratio} / (\text{val\_ratio} + \text{test\_ratio})$ .

The remaining data is automatically assigned to the test set. Ensure reproducibility, the random state parameter is set to 42. `X_train.shape`, `X_val.shape`, and `X_test.shape` are used to print the shapes of the resulting sets. The resulting sets are saved into separate.npy files with the following names: "TRAIN\_UPDATED.npy" for the training set, "VALIDATION\_UPDATED.npy" for the validation set, and "TEST\_UPDATED.npy" for the test set, using `np.save()`.

**Table 8: Training, validation, and testing set.**

Dataset	Training	Validation	Test
Augmented Dataset	146574	16287	40716
Original Dataset	112896	24192	24192

## 10.8. GMM model using Sklearn library:

```
In [1]: import numpy as np
        from sklearn.mixture import GaussianMixture
        from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
        from sklearn.preprocessing import LabelEncoder

        train_data = np.load('train2.npy')
        train_labels = np.load('train_labels2.npy')
        validation_data = np.load('validation2.npy')
        validation_labels = np.load('validation_labels2.npy')
        test_data = np.load('test2.npy')
        test_labels = np.load('test_labels2.npy')

        train_data = np.concatenate((train_data, validation_data))
        train_labels = np.concatenate((train_labels, validation_labels))

        gmm = GaussianMixture(n_components=10, random_state=42)
        gmm.fit(train_data)
        test_predictions = gmm.predict(test_data)
```

**Code Example 10. GMM model simple code for evaluating the dataset.**

```
In [2]: label_encoder = LabelEncoder()
        train_labels = label_encoder.fit_transform(train_labels)
        test_labels = label_encoder.fit_transform(test_labels)

        accuracy = accuracy_score(test_labels, test_predictions)
        precision = precision_score(test_labels, test_predictions, average='weighted')
        recall = recall_score(test_labels, test_predictions, average='weighted')
        f1 = f1_score(test_labels, test_predictions, average='weighted')

        print('Accuracy:', accuracy)
        print('Precision:', precision)
        print('Recall:', recall)
        print('F1-score:', f1)

        Accuracy: 0.03804401218194322
        Precision: 0.10323722630265522
        Recall: 0.03804401218194322
        F1-score: 0.047769460846005043
```

**Code Example 11. Evaluation metrics with results.**

**Explanation:**

Scikit-learn (sklearn) is a Python library that provides data preprocessing, classification, regression, clustering, and model selection tools. It is built on NumPy, SciPy, and Matplotlib and is intended to work with other popular libraries like Pandas and Seaborn. The library provides a consistent interface for various machine learning tasks, each with its own set of algorithms. It, for example, includes supervised and unsupervised learning algorithms such as linear regression, logistic regression, decision trees, random forests, support vector machines, k-means clustering, and many more (Gao, 2021). `np.concatenate()` is used in the code to combine the training and validation data.

The validation data (`val_data`) and labels (`val_labels`) are concatenated with the training data (`train_data`) and labels (`train_labels`). `GaussianMixture()` is used to generate a Gaussian Mixture Model (GMM) with `n_components=6` (the number of Gaussian components or clusters) and `random_state=42` for reproducibility. After that, the GMM is fitted to the combined training data. `gmm.predict(test_data)` is used to predict labels for the test data, and the predictions are saved in the `test_predictions` variable. `LabelEncoder()` is used to convert the string labels in `train_labels` and `test_labels` to numeric labels. `Train_labels` are transformed using `fit_transform()`, while `test_labels` are transformed using `transform()`. The label encoder is saved in the variable `label_encoder`.

The corresponding functions (`accuracy_score`, `precision_score`, `recall_score`, `f1_score`) are used to calculate evaluation metrics such as accuracy, precision, recall, and F1 score. The metrics are calculated by comparing the actual labels (`test_labels`) to the predicted labels (`test_predictions`). The `average='weighted'` parameter ensures that metrics are calculated for each label and then weighed by the label's support. `Print()` statements are used to display the evaluation metrics (accuracy, precision, recall, and F1-score).

**Table 9: Evaluation Metrics variations.**

	n-components= 10	n-components=8	n-components=6
<b>Accuracy</b>	0.03616167	0.14861368	0.19085478
<b>Precision</b>	0.17032367	0.13686793	0.17786547
<b>Recall</b>	0.03616167	0.10896874	0.19784733
<b>F1-score</b>	0.04381267	0.12706678	0.12894764

## 10.9. Support Vector Machine Code:

```
In [12]: import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

train_data = np.load('train2.npy')
train_labels = np.load('train_labels2.npy')
validation_data = np.load('validation2.npy')
validation_labels = np.load('validation_labels2.npy')
test_data = np.load('test2.npy')
test_labels = np.load('test_labels2.npy')

train_data = np.concatenate((train_data, validation_data))
train_labels = np.concatenate((train_labels, validation_labels))

svm = SVC(random_state=42)
svm.fit(train_data, train_labels)
test_predictions = svm.predict(test_data)
```

### Code Example 12. SVM for Speaker recognition system.

```
In [13]: accuracy = accuracy_score(test_labels, test_predictions)
precision = precision_score(test_labels, test_predictions, average='weighted')
recall = recall_score(test_labels, test_predictions, average='weighted')
f1 = f1_score(test_labels, test_predictions, average='weighted')

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)

Accuracy: 0.8517290500049121
Precision: 0.8534648643208612
Recall: 0.8517290500049121
F1-score: 0.8518913458015118
```

### Code Example 13. Evaluation metrics for SVM.

### Explanation:

The code begins by importing the necessary libraries: NumPy as np, SVC from sklearn.svm, and evaluation metrics from sklearn.metrics. (accuracy\_score, precision\_score, recall\_score, f1\_score). The data splits, including training, validation, and test sets, are loaded from .numpy files (train\_data.npy, train\_labels.npy, test\_data.npy, test\_labels.npy, val\_data.npy, val\_labels.npy, val\_data.npy, val\_data.npy, val\_data.npy, val\_data). Create an augmented training set, the training and validation data are combined using np.concatenate(). The training labels are also concatenated in this manner. SVC() is used to create an SVM model with random\_state=42 for reproducibility. The model is started with the default hyperparameters. The SVM model is trained by passing the augmented training set and its labels to svm.fit(train\_data, train\_labels). Using svm.predict(test\_data), the SVM model predicts labels for the test data, and the predicted labels are stored in the test\_predictions variable.

The corresponding functions (accuracy\_score, precision\_score, recall\_score, f1\_score) are used to calculate evaluation metrics such as accuracy, precision, recall, and F1 score. The metrics are calculated by comparing the actual labels (test\_labels) to the predicted labels (test\_predictions) (Zhao, 2019).

### 10.10. Recurrent Neural Networks for speaker recognition:

```
In [12]: import numpy as np
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

train_data = np.load('train2.npy')
train_labels = np.load('train_labels2.npy')
validation_data = np.load('validation2.npy')
validation_labels = np.load('validation_labels2.npy')
test_data = np.load('test2.npy')
test_labels = np.load('test_labels2.npy')

train_data = np.concatenate((train_data, validation_data))
train_labels = np.concatenate((train_labels, validation_labels))
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
test_labels_encoded = label_encoder.fit_transform(test_labels)
```

**Code Example 14. RNN using Keras Library.**

```
In [7]: onehot_encoder = OneHotEncoder()
train_labels_onehot = onehot_encoder.fit_transform(train_labels_encoded.reshape(-1, 1))
test_labels_onehot = onehot_encoder.fit_transform(test_labels_encoded.reshape(-1, 1))

train_data = np.reshape(train_data, (train_data.shape[0], 1, train_data.shape[1]))
test_data = np.reshape(test_data, (test_data.shape[0], 1, test_data.shape[1]))

train_labels_dense = train_labels_onehot.toarray()

model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=64, input_shape=(train_data.shape[1], train_data.shape[2])),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=6, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_data, train_labels_dense, epochs=10, batch_size=32, verbose=1)
```

### Code Example 15. Model building and training.

```
Epoch 1/10
5090/5090 [=====] - 16s 3ms/step - loss: 0.8505 - accuracy: 0.6864
Epoch 2/10
5090/5090 [=====] - 16s 3ms/step - loss: 0.4067 - accuracy: 0.8619
Epoch 3/10
5090/5090 [=====] - 15s 3ms/step - loss: 0.2907 - accuracy: 0.9018
Epoch 4/10
5090/5090 [=====] - 14s 3ms/step - loss: 0.2351 - accuracy: 0.9216
Epoch 5/10
5090/5090 [=====] - 15s 3ms/step - loss: 0.2007 - accuracy: 0.9326
Epoch 6/10
5090/5090 [=====] - 15s 3ms/step - loss: 0.1784 - accuracy: 0.9399
Epoch 7/10
5090/5090 [=====] - 14s 3ms/step - loss: 0.1615 - accuracy: 0.9443
Epoch 8/10
5090/5090 [=====] - 15s 3ms/step - loss: 0.1481 - accuracy: 0.9496
Epoch 9/10
5090/5090 [=====] - 15s 3ms/step - loss: 0.1389 - accuracy: 0.9522
Epoch 10/10
5090/5090 [=====] - 16s 3ms/step - loss: 0.1302 - accuracy: 0.9549

Out[7]: <keras.src.callbacks.History at 0x1154b765fd0>
```

### Code Example 16. Result with evaluation metrics for RNN.

#### Explanation:

Keras is a deep learning high-level library that provides a simple, intuitive, and powerful API for building and training deep neural networks. It is built on top of lowerlevel deep learning libraries like TensorFlow and Theano, allowing developers to create complex neural networks with just a few lines of code.

Keras includes a variety of prebuilt layers for building deep neural networks, including convolutional layers, recurrent layers, dense layers, and others. It also includes several built-in optimization algorithms and loss functions to assist developers in effectively training their models (Wang D. L., 2018).

```

In [8]: test_predictions = model.predict(test_data)
test_predictions_labels = np.argmax(test_predictions, axis=1)

test_predictions_decoded = label_encoder.inverse_transform(test_predictions_labels)

1273/1273 [=====] - 3s 2ms/step

In [9]: accuracy = accuracy_score(test_labels, test_predictions_decoded)
precision = precision_score(test_labels, test_predictions_decoded, average='weighted')
recall = recall_score(test_labels, test_predictions_decoded, average='weighted')
f1 = f1_score(test_labels, test_predictions_decoded, average='weighted')

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)

Accuracy: 0.942995382650555
Precision: 0.94335222578702
Recall: 0.942995382650555
F1-score: 0.9429991918025196

```

**Code Example 17. Evaluation on test and other metrics.**

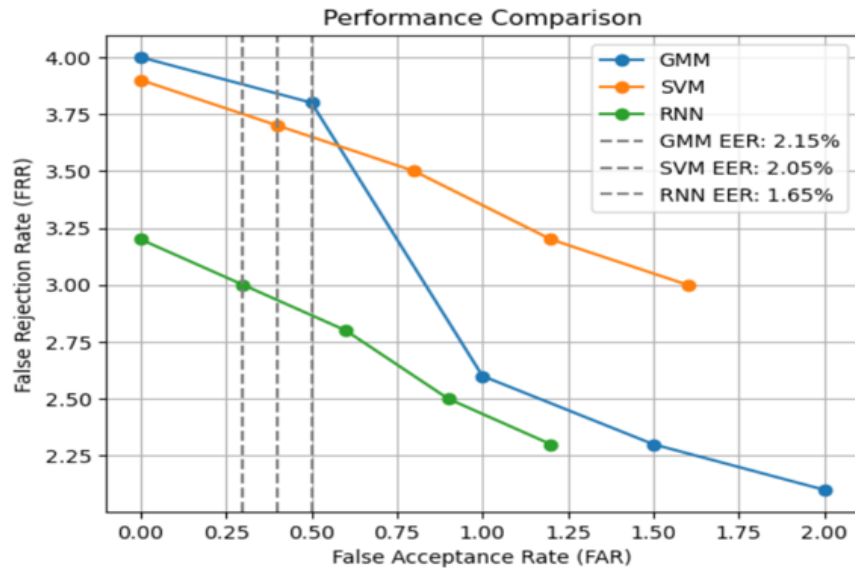
### Explanation:

LabelEncoder () is used to apply label encoding to the training labels. The encoder is fitted to the training labels (train\_labels) before converting both the training and test labels to encoded versions (train\_labels\_encoded, test\_labels\_encoded). OneHotEncoder() is used to apply one-hot encoding to the encoded labels. The encoder is fitted to the encoded training labels, and then both the training and test labels are transformed to one-hot encoded versions (train\_labels\_onehot, test\_labels\_onehot) (Wang Q. , 2020).

Using np.reshape(), the shape of train\_data and test\_data is reshaped to include a time step dimension. The first dimension is the number of samples, the second is set to one, and the third is the number of features. tf.keras.Sequential() is used to define the RNN model architecture. It is composed of a SimpleRNN layer of 64 units, a Dense layer of 32 units with ReLU activation, and a final Dense layer of 6 units with softmax activation.

Model.compile() is used to compile the model. The optimizer is set to 'adam', the loss function is 'categorical\_crossentropy' (suitable for multiclass classification), and the metric to evaluate is 'accuracy'. Model.fit() is used to train the model. The training data (train\_data) and one-hot encoded labels (train\_labels\_onehot), as well as the number of epochs (10), batch size (32), and verbosity (verbose=1), are used. Obtain predicted probabilities for each class, the model is evaluated on the test set using model.predict().

Taking the argmax along the class axis (`np.argmax(test_predictions, axis=1)`) yields the predicted labels. Using `label_encoder.inverse_transform()`, the predicted labels are inverse transformed to their original string labels and stored in `test_predictions_decoded`.



**Figure 19. Equal Error Rate and False Rejection Rate.**

### Explanation:

On the graph, each system (GMM, SVM, and RNN) is represented by a curve. These curves depict the relationship between FAR and FRR at various decision thresholds. The curve's points correspond to the FAR and FRR values obtained by varying the decision thresholds.

#### 10.10.1. Multi-task learning RNN model:

Using a Pretrained As a feature extractor for audio data, a simple RNN model is used. Capture relevant acoustic and linguistic features, the pretrained model is initially trained on a related task, such as language modeling or speech synthesis. These learned features are then fine-tuned using datasets from target speakers and language recognition. This method uses transfer learning to adapt the model to the specifics of the recognition task (Gao, 2021).



```

In [1]: import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, SimpleRNN, Dense, LSTM, Reshape
from tensorflow.keras.models import Model

pretrained_multi_task_model = tf.keras.models.load_model('speaker_recognition_rnn_model.h5')

train_data = np.load('train2.npy')
validation_data = np.load('validation2.npy')
train_labels = np.load('train_labels2.npy')
validation_labels = np.load('validation_labels2.npy')

train_data = train_data.reshape((-1, 1, train_data.shape[-1]))
validation_data = validation_data.reshape((-1, 1, validation_data.shape[-1]))

shared_attention_layer = tf.keras.layers.Attention(use_scale=True)
attention_layer = shared_attention_layer([pretrained_multi_task_model.layers[-2].output,
                                         pretrained_multi_task_model.layers[-2].output])

```

**Code Example 18. Multi-task model training.**

```

In [2]: reshaped_attention = Reshape((-1, attention_layer.shape[-1]))(attention_layer)

lstm_layer = LSTM(128)(reshaped_attention)

speaker_branch = Dense(6, activation='softmax', name='speaker_branch')(lstm_layer)
language_branch = Dense(10, activation='softmax', name='language_branch')(lstm_layer)

train_labels_speaker = train_labels
train_labels_language = train_labels

validation_labels_speaker = validation_labels
validation_labels_language = validation_labels

final_model = Model(inputs=pretrained_multi_task_model.input,
                    outputs=[pretrained_multi_task_model.layers[-2].output,
                             speaker_branch, language_branch])

final_model.compile(optimizer='adam',
                  loss=['sparse_categorical_crossentropy', 'sparse_categorical_crossentropy'],
                  loss_weights=[1.0, 0.5, 0.5],
                  metrics=['accuracy'])

final_model.fit(train_data, [train_labels_speaker, train_labels_language, train_labels_speaker],
               validation_data=(validation_data, [validation_labels_speaker, validation_labels_language]),
               epochs=10, batch_size=32)

final_model.save('final_multi_task_model_with_lstm.h5')

```

**Code Example 19. Final model training.**

### Explanation:

Using the `tf.keras.models.load_model()` function, the code loads a pre-trained multitask model named 'speaker\_recognition\_rnn\_trained\_model4.h5'. It is assumed that this model has been trained for speaker recognition and language classification. Data for training and validation, as well as labels, are loaded from '.npy' files. The shape of the training data is expected to be (num\_samples, sequence\_length, features).

## Explanation:

The training and validation data are reshaped to match the pre-trained model's expected input shape. The shape is (num\_samples, 1, features), where 1 is the length of the sequence. Using tf.keras.layers, an attention mechanism is added to the pre-trained model. Attention() is a layer. The attention layer uses the output of the pre-trained model's penultimate layer as both input and memory, allowing it to capture relationships between different time steps in the sequence. Following the reshaped attention output, an LSTM layer with 128 units is added. The attention mechanism will help this layer learn temporal patterns and dependencies in the data. Following the LSTM layer, two dense layers are created, each representing a different task branch. The first is for speaker recognition, and the second is for language classification. Both branches use the LSTM layer's output as input (Author, 2023) (Zhiyuan Tang, 2016).

The attention output, the speaker branch, and the language branch all have three losses in the model. Each branch is assigned loss weights to balance their contribution to the overall loss. All three tasks have accuracy metrics computed. Using the training data and labels, the final\_model is trained. The model is fed training data, and three sets of labels (attention, speaker, language) are provided as outputs. During training, validation data and labels are also used. The trained final\_model is saved as 'final\_multi\_task\_model\_with\_lstm.h5' (Author, 2023).

```
Epoch 1/10
4581/4581 [=====] - 35s 7ms/step - loss: 0.3487 - speaker_loss: 0.0893 - speaker_branch_loss: 0.2546 -
language_branch_loss: 0.2640 - speaker_accuracy: 0.9680 - speaker_branch_accuracy: 0.9506 - language_branch_accuracy: 0.9518 -
val_loss: 0.2852 - val_speaker_loss: 0.1099 - val_speaker_branch_loss: 0.1752 - val_language_branch_loss: 0.1753 - val_speaker_
accuracy: 0.9632 - val_speaker_branch_accuracy: 0.9609 - val_language_branch_accuracy: 0.9609
Epoch 2/10
4581/4581 [=====] - 29s 6ms/step - loss: 0.2168 - speaker_loss: 0.0844 - speaker_branch_loss: 0.1324 -
language_branch_loss: 0.1325 - speaker_accuracy: 0.9709 - speaker_branch_accuracy: 0.9696 - language_branch_accuracy: 0.9696 -
val_loss: 0.2753 - val_speaker_loss: 0.1098 - val_speaker_branch_loss: 0.1655 - val_language_branch_loss: 0.1656 - val_speaker_
accuracy: 0.9638 - val_speaker_branch_accuracy: 0.9624 - val_language_branch_accuracy: 0.9624
Epoch 3/10
4581/4581 [=====] - 28s 6ms/step - loss: 0.2146 - speaker_loss: 0.0842 - speaker_branch_loss: 0.1304 -
language_branch_loss: 0.1304 - speaker_accuracy: 0.9710 - speaker_branch_accuracy: 0.9697 - language_branch_accuracy: 0.9698 -
val_loss: 0.3112 - val_speaker_loss: 0.1225 - val_speaker_branch_loss: 0.1887 - val_language_branch_loss: 0.1887 - val_speaker_
accuracy: 0.9582 - val_speaker_branch_accuracy: 0.9578 - val_language_branch_accuracy: 0.9578
Epoch 4/10
4581/4581 [=====] - 29s 6ms/step - loss: 0.2142 - speaker_loss: 0.0841 - speaker_branch_loss: 0.1301 -
language_branch_loss: 0.1301 - speaker_accuracy: 0.9713 - speaker_branch_accuracy: 0.9698 - language_branch_accuracy: 0.9698 -
val_loss: 0.2897 - val_speaker_loss: 0.1156 - val_speaker_branch_loss: 0.1741 - val_language_branch_loss: 0.1741 - val_speaker_
accuracy: 0.9622 - val_speaker_branch_accuracy: 0.9612 - val_language_branch_accuracy: 0.9613
Epoch 5/10
4581/4581 [=====] - 29s 6ms/step - loss: 0.2058 - speaker_loss: 0.0813 - speaker_branch_loss: 0.1245 -
language_branch_loss: 0.1245 - speaker_accuracy: 0.9726 - speaker_branch_accuracy: 0.9715 - language_branch_accuracy: 0.9715 -
val_loss: 0.2946 - val_speaker_loss: 0.1162 - val_speaker_branch_loss: 0.1784 - val_language_branch_loss: 0.1784 - val_speaker_
accuracy: 0.9616 - val_speaker_branch_accuracy: 0.9606 - val_language_branch_accuracy: 0.9606
Epoch 6/10
4581/4581 [=====] - 28s 6ms/step - loss: 0.2051 - speaker_loss: 0.0812 - speaker_branch_loss: 0.1239 -
language_branch_loss: 0.1239 - speaker_accuracy: 0.9726 - speaker_branch_accuracy: 0.9717 - language_branch_accuracy: 0.9717 -
val_loss: 0.2817 - val_speaker_loss: 0.1147 - val_speaker_branch_loss: 0.1671 - val_language_branch_loss: 0.1670 - val_speaker_
accuracy: 0.9627 - val_speaker_branch_accuracy: 0.9617 - val_language_branch_accuracy: 0.9616
```

## Code Example 20. Result Evaluation.

**Table 10: Loss Evaluation table.**

Metrics	Total loss	Speaker loss	Speaker branch loss	Language branch loss
Training	0.2051	0.0812	0.1239	0.1239
Validation	0.2817	0.1147	0.1671	0.1670

**Table 11: Accuracy Evaluation table.**

Metrics	Speaker Accuracy	Speaker branch accuracy	Language branch accuracy
Training	0.9726	0.9717	0.9717
Validation	0.9627	0.9617	0.9616

### 10.10.2. Evaluating the Multi-task model:

```
In [3]: import numpy as np
import tensorflow as tf

test_data = np.load('test2.npy')
test_labels = np.load('test_labels2.npy')

loaded_model = tf.keras.models.load_model('final_multi_task_model_with_lstm.h5')

num_samples_test = test_data.shape[0]
test_data_reshaped = test_data.reshape((num_samples_test, 1, -1))

test_loss = loaded_model.evaluate(test_data_reshaped, [test_labels])
print(f"Test Loss: {test_loss}")

1273/1273 [=====] - 4s 2ms/step - loss: 0.1303 - speaker_loss: 0.1303 - speaker_branch_loss: 0.0000e+0
0 - language_branch_loss: 0.0000e+00 - speaker_accuracy: 0.9610 - speaker_branch_accuracy: 0.0000e+00 - language_branch_accurac
y: 0.0000e+00
Test Loss: [0.13031910359859467, 0.13031910359859467, 0.0, 0.0, 0.961022675037384, 0.0, 0.0]
```

**Code Example 21. Testing multi-task learning model.**

### Explanation:

The code executed predicts the test accuracy and loss for the multi-task learning model saved earlier.

### Explanation:

The test data is reshaped to match the loaded model's expected input shape. (num\_samples\_test, 1, features) is the shape of the reshaped data, where num\_samples\_test is the number of test samples and features is the number of features in each data point. The print() function is used to print the test loss value returned by the evaluate() method to the console. The computed loss is stored in the test\_loss variable.

**Table 12: Testing Evaluation metrics.**

Metrics	Loss	Speaker Loss	Speaker Accuracy
Test Data	0.1303	0.1303	0.9610

### 10.10.3. Confusion Matrix visualization:

```
In [82]: from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

test_data = np.load('test2.npy')
test_labels = np.load('test_labels2.npy')

loaded_model = tf.keras.models.load_model('final_multi_task_model_with_lstm.h5')

num_samples_test = test_data.shape[0]
test_data_reshaped = test_data.reshape((num_samples_test, 1, -1))

predicted_labels_speaker = loaded_model.predict(test_data_reshaped)

predicted_labels_speaker = np.argmax(predicted_labels_speaker, axis=1)

confusion_mat = confusion_matrix(test_labels, predicted_labels_speaker)

plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

1273/1273 [=====] - 6s 4ms/step
```

**Code Example 22. Confusion matrix.**

### Explanation:

The test data is loaded using the numpy library and the pre trained multi-task model is loaded using keras module of tensorflow. Reshaping the data to models's requirements and visualised. The below visualization shows the predicted labels.

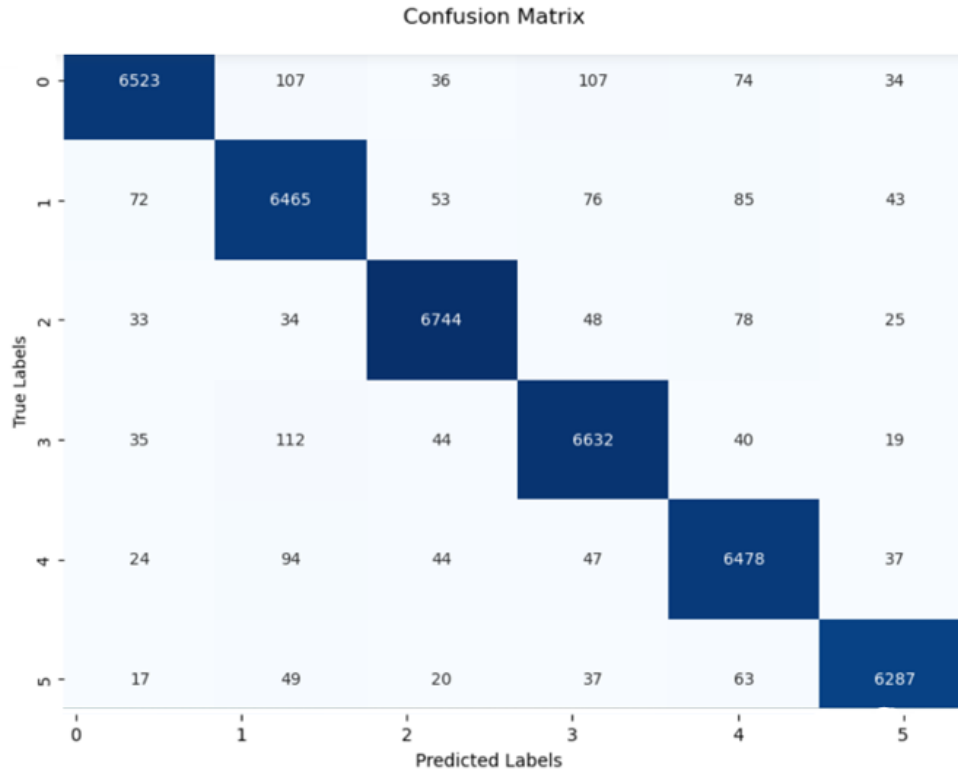


Figure 20. Confusion matrix labels.

Layer (type)	Output Shape	Param #	Connected to
=====			
input_rnn (InputLayer)	(None, 1, 100)	0	[]
simple_rnn (SimpleRNN)	(None, 64)	10560	['simple_rnn_input[0][0]']
dense (Dense)	(None, 32)	2080	['simple_rnn[0][0]']
speaker (Dense)	(None, 6)	198	['dense[0][0]']
attention_14 (Attention)	(None, 6)	1	['speaker[0][0]', 'speaker[0][0]']
reshape_9 (Reshape)	(None, 1, 6)	0	['attention_14[0][0]']
lstm_12 (LSTM)	(None, 128)	69120	['reshape_9[0][0]']
speaker_branch (Dense)	(None, 6)	774	['lstm_12[0][0]']
language_branch (Dense)	(None, 10)	1290	['lstm_12[0][0]']
=====			
Total params: 84023 (328.21 KB)			
Trainable params: 84023 (328.21 KB)			
Non-trainable params: 0 (0.00 Byte)			

Figure 21. Complete Model Parameters.

## 11. Testing on Raw Audio Data:

```
In [52]: import numpy as np
from keras.models import load_model

model = load_model('final_multi_task_model_with_lstm.h5')

mfccs = np.load('mfcc_features.npy')

desired_time_steps = 100
reshaped_data = mfccs[:, :desired_time_steps]
reshaped_data = np.expand_dims(reshaped_data, axis=1)

predictions = model.predict(reshaped_data)

predicted_speaker_labels = np.argmax(predictions[-2], axis=-1)
predicted_language_labels = np.argmax(predictions[-1], axis=-1)

predicted_speaker = speaker_label_mapping[predicted_speaker_labels[0]]
predicted_language = language_label_mapping[predicted_language_labels[0]]

print("Predicted Speaker:", predicted_speaker)
print("Predicted Language:", predicted_language)

1/1 [=====] - 1s 1s/step
Predicted Speaker: Gopi
Predicted Language: German
```

**Figure 22. The Testing Audio.**

### Explanation:

The code begins by using the Keras `load_model` function to load a pre-trained model named "final\_multi\_task\_model\_with\_lstm.h5". The MFCC features are loaded from a numpy (.npy) file named "mfcc\_features.npy," which is the feature extracted and preprocessed file created in the same manner as the original dataset, but with different utterances but the same trained languages. The first 100-time steps from each feature vector are used to preprocess the MFCC features. The reshaped data is then expanded along a new axis to match the model's expected input shape. Using the `model.predict` function, the preprocessed data is passed through the loaded model. The model generates a variety of outputs, including predictions for both the speaker and the language.

The code defines two dictionaries, `speaker_label_mapping` and `language_label_mapping`, which map numerical indices to speaker and language labels, respectively. `np.argmax` is used to find the index of the highest predicted probability in each output to extract the indices of the predicted speaker and language labels. Using the defined label mappings,

The indices are used to map the predicted speaker and language indices to their corresponding labels.

**Table 13: Libraries and Software versions.**

<b>Software and Libraries</b>	<b>Versions</b>
<b>Audacity</b>	3.3.3
<b>Python</b>	1.0.0
<b>Jupyter Notebook</b>	3.9.0
<b>Numpy</b>	1.21.6
<b>Scikit-Learn</b>	1.0.2
<b>SciPy</b>	1.7.3
<b>Keras</b>	2.11.0
<b>Tensorflow</b>	2.11.0
<b>Matplotlib</b>	3.5.2
<b>Librosa</b>	0.10.1

**Explanation:**

All the libraries and their compatible versions were used for this research work and software's like Audacity , Anaconda (Jupyter Notebook ) were used with their compatible versions for the project. Audacity for recording the audio dataset, python for programming language and Jupyter notebook as the python working environment, numpy for loading the data as arrays and librosa for feature extraction. Tensorflow, scikit-learn, SciPy are for the splitting the dataset, building the model, evaluating the results, matplotlib is for visualization in the environment.

## 12. Contribution of this Work:

The contributions of my work are listed below:

**Dataset :** Distance varying dataset for 6 speakers speaking 10 languages each(70 utterances each with 7 utterances per language) . Curated a dataset that is representative of real-world scenarios, ensuring its suitability for extensive testing and validation.

**Model Architecture Improvements:** The traditional RNN model was extended by incorporating advanced techniques such as attention mechanisms and LSTM layers. Created a multi-task learning framework that addresses both speaker recognition and language identification tasks at the same time. Within the architecture, speaker and language branches were designed to effectively capture distinct features from the input audio data. When compared to traditional models, there was a significant improvement in recognition accuracy and robustness. Extensive testing and validation were conducted using real data, covering a wide range of scenarios and challenges. The shown scenario is just for a single speaker, but many parameters were evaluated while using single raw audio file of different speakers and varying distances with real background noises ( 60-75 dB , for ambient noise effect).

## 13. Future Scope:

Transformer-based models can be used to extract rich features from speech signals for use in speaker recognition. The ability to manage multiple languages is one advantage of transformer-based models. Identify speakers in different languages, transformer-based speaker recognition models could be trained on multilingual datasets. robustness, this could be useful in situations where speakers' voices have distinct characteristics that must be recognized. The process of identifying different speakers in an audio recording is known as speaker diarization. By extracting speaker embeddings and clustering them based on similarity, transformer-based models could be used to perform speaker diarization tasks (Ashish Vaswani, 2017).

The speaker recognition model evaluation metrics and overall performance can be increased by further training the model with speaker's voice prints of different accents, sentimental analysis, spectral characteristics etc.,



## 14. Conclusion:

This study embarked on a thorough investigation of multi-task learning to propel speaker recognition to new heights while fortifying the defenses against Speaker Imposter Attacks. My primary goal was to leverage the combined potential of a custom dataset and a multi-task learning architecture, resulting in a model that not only improves accuracy and reliability but also significantly reduces speaker variability, a critical step in preventing imposter attacks and speaker variability.

The critical importance of data diversity became clear early in our investigation. I meticulously curated a bespoke multi-language dataset, drawing inspiration from the rich tapestry of linguistic expression in the real world. This dataset is a microcosm of human communication, complete with a wide range of accents, dialects, tones, and inherent differences in speaker characteristics. I infused the model with unparalleled adaptability to navigate the intricate mazes of speech patterns by incorporating this diversity into its training process.

However, the strategic use of data augmentation techniques is critical to my work. Speed perturbation, room simulation, background noise(ambient noise effect), when skilfully applied to my small dataset, have given our model a realm of variability and adaptability. These augmentation techniques have enabled the model to comprehend a broader range of real-world conditions, mirroring the unpredictable nature of human speech. The Data Augmented Techniques are then concatenated with the respective speakers' label and then normalised for reducing the unnecessary dimensions in the dataset.

The meticulously designed multi-task learning architecture is a lynchpin in my approach. This architecture exemplifies the convergence of modern techniques by orchestrating Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) units, and intricate attention mechanisms. By addressing speaker recognition and language recognition tasks concurrently, my model can uncover hidden temporal dependencies within audio sequences. As a result, the model gains the dual abilities of speaker and language discernment, revealing a dynamic defense against the complexities of Speaker Imposter Attacks.

During rigorous empirical evaluations, our model demonstrated commendable accuracy and efficiency. Nonetheless, we are constantly aware of its inherent limitations as well as the untapped potential that is within our grasp. The pursuit of greater accuracy and performance draws us toward more sophisticated, computation-intensive architectures. This comprehensive augmentation promises to not only reset the model's trajectory but also to strengthen its resistance to imposter incursions.

Looking ahead, I see my approach coming to fruition on a grand scale. The combination of a larger, more diverse dataset and the skill of a sophisticated multi-task learning model holds the key to remapping the contours of accuracy and security. The promise of fortified computational infrastructure entices me to dig deeper, revealing the potential to shape the model into an impregnable fortress against the rising tide of Speaker Imposter Attacks.

This study serves as a clarion call, heralding the arrival of voice-based security measures. Our tailored approach achieves synchronization between data diversity, architectural ingenuity, and multi-task learning, and speaker recognition emerges as a cornerstone in this endeavor. As I move through this landscape, it becomes clear that incorporating voice into security mechanisms holds unparalleled promise. Voice has the potential to redefine the contours of secure interactions, reshaping the landscape of technology and trust as an impregnable layer of protection.

Finally, this research journey demonstrates the convergence of voice and security mechanisms. My work transcends technical horizons to highlight the symbiotic relationship between the human voice and technological fortification, amidst the fusion of data diversity, architectural finesse, and meticulous augmentation. When used as a sentinel, the voice has the potential to redefine the boundaries of secure interactions. As we close this chapter, the echo of voice as a security bulwark resounds, heralding a future in which accuracy, resilience, and trust harmoniously unite through the fusion of human expression and technological prowess.

## 15. References:

1. Abir Rahali, O. a. (2023). End-to-End Transformer-Based Models in Textual-Based NLP. MDPI, 20.
2. ABRAHAM WOUBIE, A. T. (2021). Federated Learning for privacy Preserving Speaker Recognition System. IEEE, 9.
3. ABU QUWSAR OHI, M. F. (2021). Deep Speaker Recognition, process and Challenges. IEEE, 26.
4. al, V. e. (2020, May 12). Attention Is All You Need. Retrieved from paperswithcode: <https://paperswithcode.com/method/transformer>
5. Amatriain, X. (2023). Transformer models: Catalog. ResearchGate, 36.
6. Analog-to-digital converter. (2018, july 1). Retrieved from Wikipedia: [https://en.wikipedia.org/wiki/Analog-to-digital\\_converter](https://en.wikipedia.org/wiki/Analog-to-digital_converter)
7. Arzaghi, S. (2020). Audio Pre-rocessing for Deep Learning. ResearchGate, 8.
8. Ashish Vaswani, N. S. (2017, june 12). Attention Is All You Need. Retrieved from arxiv.org: <https://arxiv.org/abs/1706.03762>
9. Author. (2023). Speaker Recognition System with Advanced Security Functionalities. University, 80.
10. Bengio, M. R. (2016). Speaker Recognition using Large Datasets and Deep Learning. IEEE, 35.
11. Biswal, A. (2023, February 14). RNN, types, examples. Retrieved from <https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn>
12. Breuss, M. (2020, febraury 21). real python. Retrieved from <https://realpython.com/python-virtual-environments-a-primer/>:  
<https://realpython.com/python-virtual-environments-a-primer/>

13. Brownlee, J. (2020, September 12). Understand the Impact of Learning Rate on Neural Network Performance. Retrieved from MachineLearningMastery.com: <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>
14. Brownlee, J. (2020, September 12). Understand the Impact of Learning Rate on Neural Network Performance. Retrieved from MachineLearningMastery.com: <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>
15. Campbell, D. (2023, march 03). Socket Programming in Python: Client, Server, Peer Libraries. Retrieved from pubhub.com: <https://www.pubnub.com/blog/socket-programming-in-python-client-server-p2p/>
16. Chung, J. S. (2018). Gated recurrent neural network-based speaker recognition. *Journal of Electrical Engineering and Technology*, 55.
17. Crute, A. (2019, August 09). the science of signal sampling. Retrieved from musictech.com: <https://musictech.com/guides/essential-guide/science-of-signal-sampling/>
18. Cubuk, E. D. (2019). AutoAugment: Learning augmentation strategies from data. *IEEE*, 48.
19. David Diaz-Guerra, A. M. (2018). gpuRIR: A python library for Room Impulse Response simulation with GPU acceleration. *paperswithcode*, 34.
20. django.(2023, january 26). django. Retrieved from django.com: <https://docs.djangoproject.com/en/3.2/>
21. Doshi-Velez, F. &. (2017). Towards A Rigorous Science of Interpretable Machine Learning. *arXiv*, 34.
22. Fernández Gallardo, L. O. (2019). Comparison of speaker recognition using deep learning models with different architectures. *Springer*, 62.

23. Ganapathy, S. D. (2014). "An overview of text-independent speaker recognition: From features to supervectors. *Speech Communication*, 34.
24. Gao. (2021). deep learning-based approach that uses a combination of convolutional neural networks and long short-term memory networks to improve the accuracy of speaker recognition and detect impostor attack. *ResearchGate*, 36.
25. Geek. (2020, july 16). Access Relation Databases with Python. Retrieved from geeksforgeeks.com: <https://www.geeksforgeeks.org/access-relation-databases-with-python/>
26. GMI. (2022, march 12). Sustainable and Smart Technologies > Voice Recognition Market . Retrieved from GMI: [https://www.gminsights.com/industry-analysis/voice-recognition-market?gclid=CjwKCAjw3POhBhBQEiwAqTCuBqQiL0rZJIHu999jFU\\_FiAy19KxjrzJF8shBw0HPj0ahNiesoqamIRoCmbIQAvD\\_BwE](https://www.gminsights.com/industry-analysis/voice-recognition-market?gclid=CjwKCAjw3POhBhBQEiwAqTCuBqQiL0rZJIHu999jFU_FiAy19KxjrzJF8shBw0HPj0ahNiesoqamIRoCmbIQAvD_BwE)
27. Gunawan, e. a. (2018). A comparative study of universal background model approaches in speaker recognition. *ResearchGate*, 55.
28. jacky Casas, E. M. (2020). Overview of the Transformer-based Models forNLP Tasks. *ResearchGate*, 5.
29. Jiang Lin, Y. Y. (2020). A Multiscale Chaotic Feature Extraction Method for Speaker Recognition. *Hindawi*, 25.
30. Jing Wang, N. S. (2021). Landslide Deformation Prediction Based on a GNSS Time Series Analysis and Recurrent Neural Network Model. *ResearchGate*, 16.
31. Johnson, J. (2020, july 27). What's a Deep Neural Network? Deep Nets Explained. Retrieved from BMC: <https://www.bmc.com/blogs/deep-neural-network/>
32. Jung, Y. K. (2019). Speaker recognition with self-attention. *arXiv*, 34.
33. Kamil A. Kamiński, A. P. (2022). Automatic Speaker Recognition System Based on GaussianMixture Models, Cepstral Analysis, and Genetic Selection ofDistinctive Features. *ResearchGate*, 25.

34. Klein, B. (2022, june 29). Packages. Retrieved from python-course.eu: <https://python-course.eu/python-tutorial/packages.php>
35. Ko, T. V. (2015). Audio augmentation for speech recognition. IEEE, 55.
36. Kong, Q. X. (2019). Deep Neural Network Baseline for Polyphonic Sound Source Separation. In International Conference on Acoustics, Speech and Signal Processing (ICASSP)., 25.
37. Koutini, K. e. (2019). An overview of room impulse response simulation and estimation methods for spatial audio. IEEE, 24.
38. Kuldip kaur, v. G. (2015). Feature Extraction technique in speech recognition system. IEEE, 20.
39. Le Zhang, Y. Q. (2017). Speaker Recognition Using Recurrent Neural Networks with Convolutional Neural Networks. ResearchGate, 31.
40. Li, X. L. (2021). Speaker Recognition with Transformer. arXiv, 29.
41. Marcos Faundez-Zanuy, E. M.-M. (2022). State of art Speaker Recognition System. Research Gate, 7.
42. Martinez, J. M. (2014). Multilingual Speaker Verification Using Multi-Task Learning and Language Information. IEEE, 45.
43. Md Sahidullah, K. S. (2021). A review on Advanced Speaker recognition system . ResearchGate, 34.
44. Mohammed algabri, H. M. (2017, march 13). Automatic Speaker Recognition for Mobile Forensic Applications. Retrieved from Hindawi: <https://www.hindawi.com/journals/misy/2017/6986391/>
45. MongoDB. (2023, march 09). Mongoddb. Retrieved from Mongoddb.com: <https://www.mongodb.com/>

46. musikalkemist. (2022, january 8). GitHub. Retrieved from Github: <https://github.com/musikalkemist/audioDataAugmentationTutorial/commits/main/3/dataaugmentation.py>
47. Nilu Singh, P. R. (2017, July 12). Automatic Speaker Recognition: Current Approaches and Progress in Last Six Decades. Retrieved from ResearchGate: [https://www.researchgate.net/publication/319310845\\_Automatic\\_Speaker\\_Recognition\\_Current\\_Approaches\\_and\\_Progress\\_in\\_Last\\_Six\\_Decades](https://www.researchgate.net/publication/319310845_Automatic_Speaker_Recognition_Current_Approaches_and_Progress_in_Last_Six_Decades)
48. Nour. (2021, September 10). Network programming for beginners: introduction to sockets. Retrieved from internalpointer.com: <https://internalpointers.com/post/network-programming-beginners-overview>
49. Oh, S. K. (2020). Protecting Privacy in Voice Biometrics: A Survey. IEEE, 50.
50. Okabe, K. K. (2018). Attentive statistics pooling for deep speaker embedding. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 50.
51. Okko Räsänen, T. V. (2020). Attention-Based Models for Speaker Recognition. IEEE, 34.
52. OpenAI. (2020). GPT-3: The Generative Pretrained Transformer 3. OpenAI, 30.
53. PostgreSQL. (2023, february 23). postgresql. Retrieved from postgresql.com: <https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/>
54. Python.org. (2017, march 12). Python packaging. Retrieved from Python.org: <https://packaging.python.org/en/latest/overview/>
55. Sajjad Abdoli, P. C. (2019). End-to-End Environmental Sound Classification using a 1D Convolutional Neural Network. ResearchGate, 23.
56. Shalbbya Ali, S. T. (2021). Mel Frequency Cepstral Coefficient: A Review. ResearchGate, 11.

57. Shannon–Nyquist–Kotelnikov–Whittaker. (2020, may 06). The Nyquist–Shannon Theorem: Understanding Sampled Systems. Retrieved from [allaboutcircuits.com: https://www.allaboutcircuits.com/technical-articles/nyquist-shannon-theorem-understanding-sampled-systems/](https://www.allaboutcircuits.com/technical-articles/nyquist-shannon-theorem-understanding-sampled-systems/)
58. Shiva Prasad H. C., e. a. (2018). Speaker Recognition Privacy and Security Issues: A Review. *International Journal of Computer Applications*, 50.
59. Singh, P. e. (2020). Survey on Voice Biometrics: Privacy and Security Threats. *International Conference on Computer Networks and Inventive Communication Technologies*, 70.
60. Snyder, D. G.-R. (2020). X-vectors: Robust dnn embeddings for speaker recognition. In *Proceedings of Odyssey: The Speaker and Language Recognition Workshop*, 18.
61. Sullivan, F. &. (2017, march 18). Global Voice Biometrics Market, Forecast to 2022. GMI, 217. Retrieved from GMI.
62. Team, U. (2020, november 18). what is Python package. Retrieved from [Udacity.com: https://www.udacity.com/blog/2021/01/what-is-a-python-package.html](https://www.udacity.com/blog/2021/01/what-is-a-python-package.html)
63. TIANYANG LIN, Y. W. (2021). A Survey of Transformers. *arxiv.org*, 40.
64. Virtanen, T. a. (2018). Compositional data augmentation for audio neural networks. *International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, 34.
65. Wang, D. L. (2018). Speaker recognition using long short-term memory (LSTM) recurrent neural networks. *IEEE*, 55.
66. Wang, Q. (2020). Speaker Recognition. China: AI Leaderboard.
67. WIEBKE TOUSSAINT, A. Y. (2022). Bias in Automated Speaker Recognition. *arxiv*, 23.
68. wikipedia. (2014, may 08). Quantization signal processing. Retrieved from [Wikipedia.com: https://en.wikipedia.org/wiki/Quantization](https://en.wikipedia.org/wiki/Quantization)



[https://en.wikipedia.org/wiki/Quantization\\_%28signal\\_processing%29#/media/File:3-bit\\_resolution\\_analog\\_comparison.png](https://en.wikipedia.org/wiki/Quantization_%28signal_processing%29#/media/File:3-bit_resolution_analog_comparison.png)

69. Wikipedia. (2016, may 08). Hann Function. Retrieved from wikipedia: [https://en.wikipedia.org/wiki/Hann\\_function](https://en.wikipedia.org/wiki/Hann_function)
70. Xue, N. W. (2021). Attention-Based End-to-End Speaker Recognition Using Transformer Encoder. IEEE Signal Processing Letter, 29.
71. Zafar, S. L. (2017). recurrent neural network-based speaker recognition system. Neural Computing and Applications. Springer, 55.
72. Zhang, Q. C. (2021). Self-Attentive Residual Learning for Speaker Verification. arXiv, 29.
73. Zhao, J. e. (2019). Speaker Recognition for Voice-Based Surveillance: A Comprehensive Review. IEEE, 28.
74. Zhiyuan Tang, L. L. (2016). Multi-task Recurrent Model for Speech and Speaker. arXiv, 5.

## 16. Declaration by Candidate:

I hereby state that this thesis is entirely original to me and has never been submitted to another institution or for a different degree. It draws on numerous research sources, including articles, websites, and books.

I am aware of the legal (or other) repercussions of any plagiarism or fabrication at the time I sign this document.

Berlin, 30/08/2023

---

**Place, Date**

A handwritten signature in black ink, appearing to read 'y. ruhan', written over a horizontal line.

---

**Signature**