An-Najah National University

Electrical and Telecommunications Engineering

Departments

GRADUATION PROJECT 2

"Your Whisper Key-Sound ID Unlock"

Submitted in partial fulfillment of the requirements for Bachelor degree in Electrical and Telecommunications Engineering

Supervisor: Dr. Falah Hasan.

Students Name:

Jana Nazem Hanini (12011091)

Noor AlDeen Moneer Mahameed (12010248)

Academic year: 2024/2025

❖ DEDICATION

بسم الله الرحمن الرحيم "وَأَن لَّيْسَ لِلْإِنسَانِ إِلَّا مَا سَعَىٰ وَأَنَّ سَعْيَهُ سَوْفَ يُرَىٰ" صدق الله العظيم

إلى هذا الوَطَنِ الحَبِيبِ...

لَكَ الحُبُّ الأَوَّلُ، وَالانْتِمَاءُ الأَبَدِيُّ، وَلَكَ هٰذا الجُهْدُ ثَمَرَةُ عِلْمٍ وَسَعْيٍ، نَضَعُهُ بَيْنَ يَدَيْكَ بِكُلِّ فَخْرٍ وَوَلَاءٍ.

إلى الأَهلِ الأَعِزَّاءِ...

أَنْتُمْ الدِّعَامَةُ الأُولَى، وَالدُّعَاءُ الصَّادِقُ فِي كُلِّ لَحْظَةٍ، هٰذَا الإِنْجَازُ مِنْكُمْ وَإِلَيْكُمْ، فَأَنْتُمُ الأَصْلُ وَالامْتِدَادُ.

إلى الأَصدِقَاءِ وَرِفَاقِ الدَّرْبِ...

كُنْتُمْ النُّورَ فِي عَتْمَةِ الطَّرِيقِ، وَاليَدِ الَّتِي أَمْسَكَتْ بِنَا حِينَ كِدْنَا أَنْ نَتَعَثَّرَ، شُكْرًا لِقُلُوبِكُمْ الَّتِي كَانَتْ مَعَنَا.

إلى الأَسَاتِذَةِ الكِرَامِ...

مَنْ حَمَلُوا شُعْلَةَ العِلْمِ، وَمَشَوْا بِهَا نَحْوَنَا بِكُلِّ صَبْرٍ وَنُبْلٍ، فَكُنْتُمْ الحَرْفَ الأَوَّلَ فِي كُلِّ إِنْجَازٍ نَكْتُبُهُ اليَوْمَ.

هٰذَا العَمَلُ لَا يُخْتَتَمُ بِأَسْمَائِنَا وَحْدَنَا، بَلْ يَحْمِلُ بَصَمَاتِ مَنْ أَحَبَّنَا، وَمَنْ آمَنَ بِنَا.

لَكُمْ مِنَّا كُلُّ الِامْتِنَانِ وَالتَّقْدِيرِ،

المُهَنْدِسَةُ جَنَى نَاظِم احْنِينِي

المُهَنْدِسُ نُورُ الدِّينِ مُنِير مَحَامِيد

❖DISCLAIMER

This report was written by students at the Electrical and Telecommunications Engineering Department, Faculty of Engineering, An-Najah National University. It has not been altered or corrected, other than editorial corrections, as a result of assessment and it may contain language as well as content errors. The views expressed in it together with any outcomes and recommendations are solely those of the students. An-Najah National University accepts no responsibility or liability for the consequences of this report being used for a purpose other than the purpose for which it was commissioned.

❖ACKNOWLEDGMENT

- First and foremost, we thank God for the strength, patience, and success that have brought us to this point and enabled us to complete this project to the fullest.

- We extend our deepest gratitude to our dear families, who have been a key supporter every step of the way. Without them, this achievement would not have been possible.

- We express our gratitude to our supervisor, Dr. Falah Hassan, for his time, effort, and guidance.

- We also thank all our esteemed teachers for answering our questions whenever we needed them.

- Finally, to our dear friends, who have been our greatest support. Thank you for everything.

❖TABLE OF CONTENTS

❖LIST OF FIGURES

❖LIST OF TABLES

❖LIST OF ABBREVIATIONS

MFCC	Mel Frequency Cepstral Coefficient
CNN	Convolutional Neural Network
RNN	Recursive Neural Networks
SVM	Support Vector Machine
IoT	Internet of Things
PID	Proportional Integral Derivative
3D	Three-Dimensional
ASR	Automatic Speech Recognition
NLP	Natural Language Processing
FFT	Fast Fourier Transform
DCT	Discrete Cosine Transform
ID	Identification
GMM	Gaussian Mixture Model
EM	Expectation-Maximization
MAP	Maximum A Posteriori
RAP	Relative Average Perturbation
APQ	Amplitude Perturbation Quotient
NHR	Noise-to-Harmonics Ratio
SPI	Soft Phonation Index
SNR	Signal to Noise Ratio

❖ABSTRACT

This project follows an intelligent design for an access control system based on voice recognition of specific individuals, enhancing security and facilitating access by relying on voice prints instead of traditional passwords. The system analyzes the unique voice data of authorized individuals, which is then recorded in real time. The recording is then recorded, allowing the door to open immediately if a match is found.

The system boasts stellar voice recognition accuracy, even in noisy environments. It relies on advanced algorithms that are remarkably capable of distinguishing the authorized person's voice from a range of different voices, making it suitable for use in real-world situations requiring a high level of security.

The project reflects the integration of voice control technologies and embedded systems, highlighting the significant potential of voice recognition technologies in building smart solutions and controlling devices.

❖CHAPTER 1: INTRODUCTION

1.1 Statement of the problem.

Traditional door security systems, such as physical keys, keypads, and access cards, are vulnerable to loss, theft, duplication, or unauthorized access. These methods also lack flexibility and may not provide adequate security in modern smart environments. Therefore, there is a need for a more intelligent and secure access control system that uses biometric features specifically, voice recognition to ensure that only authorized individuals can unlock and access secured areas. This project aims to address this issue by designing a voice-controlled door lock system that offers improved security, convenience, and reliability.

1.2 Objectives of the work.

The main objective of this project is to develop a secure and user-friendly voice-controlled door lock system that can identify and authenticate specific authorized users based on their voice. The detailed objectives include:

- Designing and implementing a voice recognition module capable of identifying pre-registered voices.
- Integrating the voice module with a microcontroller to control the door lock mechanism.
- Ensuring system reliability and accuracy under various environmental conditions.
- Enhancing user convenience by enabling hands-free, keyless access.

1.3 Scope of the work.

The scope of this project includes the design and implementation of a voice-controlled door locking system that recognizes specific, pre-recorded voice inputs. The system will be built using a microcontroller, a voice recognition module, and an electronic locking mechanism. The project will be limited to recognizing a limited number of authorized users, and the impact of environmental factors such as noise, echo, and polyphony will be tested within pre-defined limits.

1.4 Significance or importance of your work.

The proposed voice lock system provides a modern, secure, and contactless alternative to traditional entry methods such as keys, cards, and passwords. The project's significance lies in its reliance on voice recognition, a unique biometric feature that is difficult to replicate, enhancing security. The project also aligns with the growing demand for smart and automated systems in homes and workplaces. Additionally, this system offers a practical and easy to use solution for people with disabilities who may encounter difficulty with traditional locks. By reducing physical interaction and improving access control, the system contributes to a safer and smarter living environment.

1.5 Organization of the report.

This report is organized into six main chapters to provide a clear and systematic presentation of the project.

Chapter 1 introduces the project by outlining the problem statement, objectives, scope, significance, and the structure of the report.

Chapter 2 presents a comprehensive literature review, discussing previous work related to voice recognition, biometric systems, and the technologies used.

Chapter 3 explains the methodology, including the system design, biometric techniques, feature extraction processes, and noise handling methods.

Chapter 4 details the results and analysis, showcasing the system's performance with different users and discussing the extracted features.

Chapter 5 provides a discussion of the key findings, challenges faced, and the effectiveness of the proposed system.

Chapter 6 concludes the report with a summary of the outcomes, recommendations for improvement, and suggestions for future work. Additionally, a **SWOT analysis** is included to evaluate the strengths, weaknesses, opportunities, and threats of the project. Finally, **References** are provided to acknowledge the sources used throughout the research.

❖CHAPTER 2: LITERATURE REVIEW

voice recognition technology has gained significant attention in recent years due to its growing applications in security, smart systems, and human-machine interaction. Researchers have continuously sought to improve the accuracy, reliability, and noise tolerance of voice-based systems, particularly for access control. Traditional security methods, such as physical keys, PIN codes, and fingerprint sensors, have shown limitations in terms of convenience, vulnerability to theft, and environmental constraints. As a result, biometric systems, especially those based on voice recognition, have emerged as a promising alternative due to their non-contact nature and user-friendliness.

Previous studies have demonstrated that voice recognition can serve as a reliable biometric identifier when combined with appropriate signal processing techniques. For example, the use of Mel-Frequency Cepstral Coefficients (MFCCs) and Gaussian Mixture Models (GMMs) has been widely adopted to enhance the precision of speaker identification. However, early implementations struggled with noise interference and variations in voice due to environmental factors or user behavior. To address these issues, modern approaches integrate intelligent control systems and machine learning algorithms, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), to improve system adaptability and performance.

Moreover, the comparison between different biometric modalities such as fingerprint, facial, and voice recognition reveals that voice recognition offers unique advantages, particularly in accessibility for people with disabilities and its compatibility with hands-free smart environments. Researchers have also distinguished between speech recognition, which focuses on understanding spoken commands, and speaker recognition, which identifies the person speaking. The latter is the core focus of this project.

This project builds upon these studies by designing a practical, real-time, and speaker-specific voice-controlled door lock system, incorporating advanced noise handling, signal processing, and speaker verification techniques to ensure secure and efficient access control in both personal and professional settings.

❖CHAPTER 3: METHODOLOGY

3.1 Intelligent Control Systems

3.1.1 Definition of intelligent control systems:

An intelligent system uses artificial intelligence and intelligent learning control techniques without direct human intervention. Unlike traditional control systems (such as PID), which rely on mathematical models, intelligent control systems learn from data and adapt to the surrounding environment, even if they are subject to data uncertainty or inaccuracy.

3.1.2 Features of intelligent control systems:

- ✓ **Adaptiveness:** The ability to modify the system's behavior based on changes in the environment or inputs.
- ✓ **Flexibility:** The ability to handle nonlinear and complex systems that are difficult to model mathematically.
- ✓ **Control under uncertainty:** The ability to operate even in the presence of noise or incomplete data.
- ✓ **Self-learning:** Improves performance over time by learning from past experiences.

3.2 Biometric Systems

3.2.1 Definition of Biometric systems:

are a common commercial or behavioral technology used to identify people or verify their original identity. This process can be used for periods such as access to protected areas, smart devices, and even system centers.

3.2.2 Types of Biometric Features:

- ✓ Physical: Fingerprint, Face, Iris, DNA.
- ✓ Behavioral: Voiceprint, Signature, Gait, Keyboarding Style.

3.2.3 Comparison of Biometric Systems: Fingerprint, Face, and Voice Recognition.

Criterion	Fingerprint Recognition	Face Recognition	Voice Recognition
Accuracy	Very high	High, but affected by lighting and angle	Good, improving with modern technologies
Reliability	Reliable, but affected by cuts or dirt	Moderate, can be affected by facial changes	Moderate to high with voice filtering
Ease of Use	Requires touching a sensor	Requires camera alignment	No touch or alignment needed – just speak
Privacy	Can be faked using printed fingerprints	Can be spoofed with 3D images	More secure using unique voice frequencies
Accessibility	Limited for certain physical disabilities	Limited for visually impaired	Very suitable for people with disabilities
System Integration	Needs dedicated sensor hardware	Needs high-quality camera	Can use standard microphones
Cost	Medium	Relatively high	Low – no special hardware required
Environmental Adaptability	Affected by sweat or dust	Affected by lighting or shadows	Works in dark and varied conditions

Table 1-3: Cmparison of Biometric System.

3.2.4 Why Voice Recognition is the Best Option?

- ✓ **No Specialized Hardware Required:** Voice systems work with standard microphones in phones or computers.
- ✓ **User-Friendly:** The user simply needs to speak – no physical contact or precise positioning needed.
- ✓ **Strong Privacy:** Can use techniques like frequency matching or spoken passphrases to enhance security.
- ✓ **Highly Flexible:** Easy to integrate into smart systems, such as smart locks, voice assistants, and more.
- ✓ **Ideal for Smart Environments**: Perfect for hands-free access in smart homes, offices, and public systems.
- ✓ **Upgradable:** Can be improved using machine learning and signal processing.

3.2.5 Speech Recognition vs Speaker Recognition:

Feature	Speech Recognition	Speaker Recognition
Also Called	Automatic Speech Recognition (ASR)	Voice Biometrics or Speaker Identification
Purpose	Understand what is being said	Identify or verify who is speaking
Focus	Converts spoken words into text	Matches voice patterns to a specific person
Main Output	Text (e.g., "Open the door")	Identity (e.g., "This is Jana speaking")
Technology	Natural Language Processing (NLP), speech-to-text models	Machine learning on vocal features (tone, pitch.)
Sensitive to	Language, accent, grammar	Voice characteristics, emotion, background noise
Used With	Commands, dictation, transcription	Authentication and security systems

Table 2-3: Speech Recognition vs Speaker Recognition.

3.2.6 Applications of Speech Recognition

1. **Virtual Assistants**: Siri, Alexa, Google Assistant.
2. **Dictation & Transcription**: Turning spoken lectures or medical notes into text.
3. **Voice Interfaces**: Smart TVs, cars, or appliances responding to voice commands.
4. **Real-time Translation**: Translating spoken language into another in real time.
5. **Voice-Controlled Apps**: Hands-free texting or controlling apps while driving.

3.2.7 Applications of Speaker Recognition:

1. Security & Authentication

- Voice-based login to banking apps or secure systems.
- Smart locks that open only to specific people's voices.

2. Forensics & Law Enforcement

- Identifying speakers in recorded phone calls or surveillance.

3. Personalized Services

- In smart homes: the assistant recognizes who is speaking and adjusts preferences (music, lighting, temperature).
- In cars: auto-sets driving profiles (seat, mirrors) based on the driver's voice.

4. Call Centers & Customer Support

- Voice ID to verify customers instead of security questions.
- Fraud prevention in financial services.

3.2.8 Components of the speaker Recognition System:

1. Front-end processing - the "signal processing" part, which converts the sampled speech signal into set of feature vectors, which characterize the properties of speech that can separate different speakers. Frontend processing is performed both in training and testing phases.

2. Speaker modeling - this part performs a reduction of feature data by modeling the distributions of the feature vectors.

3. Speaker database - the speaker models are stored here.

4. Decision logic - makes the final decision about the identity of the speaker by comparing unknown feature vectors to all models in the database and selecting the best matching model.

3.3 Formants

3.3.1 Definition of the Formants:

Formants are the resonant frequencies of the human vocal tract. They appear as peaks in the speech spectrum and are critical in shaping the distinct sounds of vowels and some consonants.

3.3.2 Basic Definition:

A formant is a frequency band where acoustic energy is concentrated due to resonance in the vocal tract during speech production.

3.3.3 Acoustic & Anatomical Basis

1. The lungs push air through the vocal folds, which may vibrate to produce a glottal sound.
2. The sound then travels through the vocal tract (throat, mouth, nasal cavity).
3. The shape of the vocal tract selectively amplifies certain frequencies—these are formants.

3.3.4 The number and positions of formants depend on:

- Vocal tract length and shape
- Tongue and lip position
- Jaw opening
- Nasal cavity involvement

3.3.5 Key Formants (F1, F2, F3...)

Formant	Frequency Range (Typical Male Voice)	Acoustic Meaning
F1	300–900 Hz	Inversely related to tongue height. Low F1 = high tongue.
F2	850–2500 Hz	Related to tongue frontness High F2 = front tongue.
F3	1800–3000 Hz	Influenced by lip rounding and tongue tip position.
F4	3000+ Hz	Less often used, but helpful in speaker identification.

Table 3-3: Key Formants.

3.4 Mel-Frequency Cepstral Coefficients (MFCCs)

3.4.1 Definition of MFCC:

The human speech contains numerous discriminative features that can be used to identify speakers. Speech contains significant energy from zero frequency up to around 5 kHz. The objective of automatic speaker recognition is to extract, characterize and recognize the information about speaker identity. The property of speech signal changes markedly as a function of time. To study the spectral properties of speech, signal the concept of time varying Fourier representation is used. However, the temporal properties of speech signal such, as energy, zero crossing, correlation are assumed constant over a short period. That is its characteristics are short-time stationary. Therefore, using hamming window, Speech signal is divided into a number of blocks of short duration so that normal Fourier transform can be used.

3.4.2 Concept Summary:

- ✓ **Mel-frequency:** Mimics how humans perceive pitch (nonlinear frequency scale).
- ✓ **Cepstral:** Represents the rate of change in the spectral envelope (the shape of the spectrum).
- ✓ **Coefficients:** The actual numbers that describe the shape of the spectrum.

3.4.3 MFCC Extraction Steps:

- ✓ Convert the audio signal to the frequency spectrum using FFT.
- ✓ Map the frequency bands to the Mel scale (human auditory scale).
- ✓ Apply the logarithm to compress large variations.
- ✓ Use the Discrete Cosine Transform (DCT) to generate the final coefficients.

3.4.4 MFCCs vs Formants

Feature	MFCCs (Mel-Frequency Cepstral Coefficients)	Formants (F1, F2, F3...)
What They Represent	Overall spectral envelope of the speech signal	Resonant frequencies of the vocal tract
Derived From	Log Mel-spectrum → Discrete Cosine Transform (DCT)	Peaks in the speech spectrum
Biological Basis	Indirect: reflects vocal tract via spectrum shape	Direct: tied to anatomical vocal tract properties
Interpretability	Not directly interpretable	Highly interpretable (linked to articulation)
Used In	ASR, speaker ID, emotion recognition	Forensics, phonetics, speech pathology
Robustness	More robust to noise, distortion, and channel effects	Sensitive to recording quality and noise
Language Dependency	Generally, language-independent	More influenced by phoneme/language characteristics
Feature Dimensions	Typically, 13–39 (with deltas and acceleration)	Usually 3–5 (F1–F5)
Extraction Tools	Easy: Librosa, Kaldi, Python, MATLAB	Requires spectral analysis, LPC, or Praat
Application Speed	Fast, low computational cost	Slower, sometimes requires manual verification

Table 4-3: MFCCs vs Formants.

3.5 The pitch period

3.5.1 Definition of the pitch period:

Pitch period is the time interval between two consecutive repetitions of a sound wave produced by the vocal cords (often when pronouncing a vowel or tone), it determines the pitch of a speaker.

3.5.2 Technical Definition:

It is the time interval (in seconds or samples) between two consecutive peaks in a periodic audio signal produced by the vibration of the vocal cords.

3.5.3 The Importance of Pitch in Voice Recognition:

- **Speaker recognition:**

  Each person has a distinct pitch. Men typically have a fundamental frequency between 85–180 Hz, and women between 165–255 Hz.

- **Feature extraction:**

  Pitch period is used in some biometric systems as an additional feature with MFCC or LPC.

- **Signal classification:**

  It helps distinguish between:

  - **Voiced signals:** have a clear pitch period (such as vowels).
  - **Unvoiced signals:** do not have a regular pitch (such as f and s).

3.5.4 Approximate Pitch Period Values for Different Groups:

Group	Typical Fundamental Frequency (F_0)	Pitch Period (T_0) = 1 / F_0
Males	85 – 180 Hz	5.6 – 11.7 milliseconds
Females	165 – 255 Hz	3.9 – 6.1 milliseconds
Children	250 – 400 Hz	2.5 – 4 milliseconds
Elderly	100 – 200 Hz	5 – 10 milliseconds

Table 5-3: Approximate Pitch Period Values for Different Groups.

3.5.5 Explanation:

1. **Males:**
   - Have longer and thicker vocal cords → vibrate slower → lower frequency → longer pitch period.
2. **Females:**
   - Shorter and thinner vocal cords → vibrate faster → higher frequency → shorter pitch period.
3. **Children:**
   - Very small vocal folds → very high frequency → shortest pitch period.
   - Their voices tend to be higher-pitched.
4. **Elderly:**
   - Changes in muscles and nerves affect vocal folds.
   - Frequency often decreases in males and slightly increases in females, reducing the gap between genders.

3.6 Speaker Recognition as a Biometric Identity System: Training and Testing Phases.

Anatomical structure of the vocal tract is unique for every person and hence the voice information available in the speech signal can be used to identify the speaker. Recognizing a person by her/his voice is known as speaker recognition. Since differences in the anatomical structure are an intrinsic property of the speaker, voice comes under the category of biometric identity. Using voice for identity has several advantages. One of the major advantages is remote person authentication. Like any other pattern recognition systems, speaker recognition systems also involve two phases namely, training and testing. Training is the process of familiarizing the system with the voice characteristics of the speakers registering. Testing is the actual recognition task. The block diagram of training phase is shown in Figure 1-3. Feature vectors representing the voice characteristics of the speaker are extracted from the training utterances and are used for building the reference models. During testing, similar feature vectors are extracted from the test utterance, and the degree of their match with the reference is obtained using some matching technique. The level of match is used to arrive at the decision. The block diagram of the testing phase is given in Figure 1-3.

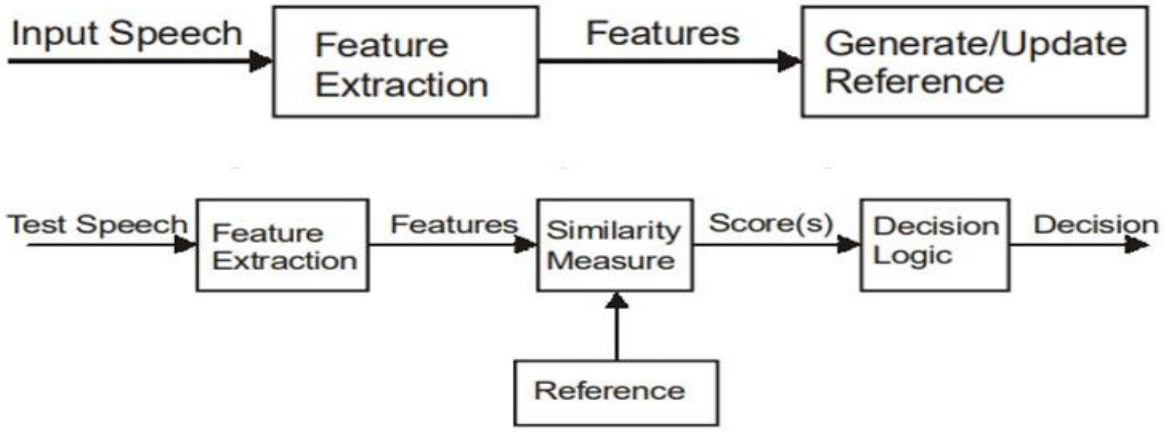


Figure 1-3: Speaker Recognition (Tranining and Testing Phases).

3.7 Noise Handling and Signal Processing Techniques

In real-world environments, voice recognition systems are often exposed to various sources of signal distortion such as background noise (side conversations, device sounds, traffic, etc.), variations in voice intensity, and changes in microphone sensitivity. To minimize the impact of these factors and improve system performance, a set of noise handling and signal processing techniques were applied.

3.7.1 Band-Pass Filtering

3.7.1.1 What is Band-Pass Filtering?

Band-pass filtering is a technique that allows frequencies within a specific range to pass through while attenuating frequencies outside this range.

3.7.1.2 Why is it Used?

- ✓ The critical frequency range for human speech typically lies between 300 Hz and 3400 Hz.
- ✓ Frequencies below 300 Hz often represent noise such as vibrations or air conditioning sounds.
- ✓ Frequenci above 3400 Hz are typically sharp noises or electrical interference.

3.7.1.3 Benefits:

- ✓ Isolates the relevant speech signals.
- ✓ Improves the signal-to-noise ratio (SNR).
- ✓ Makes the system more accurate and less sensitive to unwanted sounds.

3.7.2 Hamming Windowing

3.7.2.1 What is Hamming Windowing?

It is a mathematical function applied to small segments (frames) of the audio signal to reduce distortion when converting the signal to the frequency domain.

3.7.2.2 Why is it Used?

- ✓ The speech signal changes rapidly and cannot be accurately analyzed as a long continuous signal.
- ✓ The signal is divided into short-time frames (20 to 30 milliseconds), which can be considered stationary.
- ✓ Without windowing, abrupt changes at the frame edges cause "spectral leakage" during Fourier Transform analysis.

3.7.2.3 Benefits:

- ✓ Minimizes edge discontinuities between frames.
- ✓ Provides more accurate spectral analysis for feature extraction.

3.7.3 Signal Normalization

3.7.3.1 What is Signal Normalization?

Signal normalization is the process of adjusting the amplitude of the signal so that all samples have a consistent energy level.

3.7.3.2 Why is it Used?

- ✓ Speakers may talk closer or farther from the microphone.
- ✓ Speakers may talk loudly or softly.

3.7.3.3 Benefits:

- ✓ Ensures fair comparison across all samples regardless of voice intensity.
- ✓ Improves the performance of the recognition algorithm by focusing on the unique features of the voice rather than its volume.

3.7.4 Summary Table of Noise Handling Techniques

Technique	Purpose	Benefit
Band-Pass Filtering	Isolate human speech frequencies.	Reduces external noise.
Hamming Windowing	Prevent spectral leakage.	Accurate frequency analysis.
Signal Normalization	Standardize signal amplitude.	Fair comparison across samples.

Table 6-3: Summary of Noise Handiling Techniques.

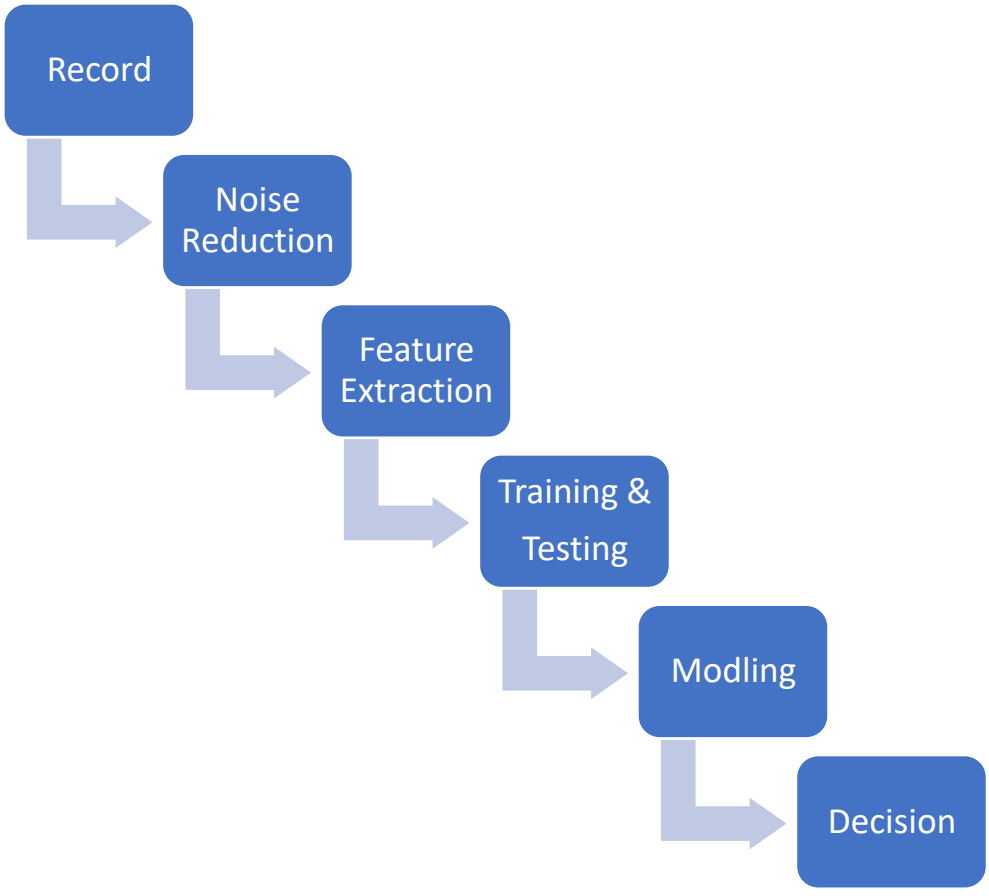


Figure 3-3: Speaker Recognition Process.

3.8 Cosine Similarity ⇨ Cosine Similarity=‖A‖‖B‖/A·B

3.8.1 What does Cosine Similarity mean in Voice Recognition?

In voice recognition, cosine similarity is used to measure how similar two voice samples are, based on their audio features, not on how loud the voice is, It checks whether two voices "sound alike" mathematically.

3.8.2 How is the voice represented?

A voice signal cannot be compared directly as raw sound.

So we first convert the voice into numerical feature vectors, such as:

- ✓ MFCC (Mel-Frequency Cepstral Coefficients) .
- ✓ Spectrogram features.
- ✓ Embeddings from CNN / Deep Learning models.

3.8.3 Why Cosine Similarity is good for Voice Recognition?

- ✓ Not affected by speaking louder or softer.
- ✓ Fast and computationally cheap.
- ✓ Works well with MFCC & deep embeddings.
- ✓ Common in speaker verification systems.

3.8.4 Limitations:

- ✓ Sensitive to background noise if features are poor.
- ✓ Requires good feature extraction.
- ✓ Threshold tuning is critical.

3.9 Mono-Spectrogram

3.9.1 What is a Mono-Spectrogram?

A mono-spectrogram is a time–frequency representation of an audio signal that has only one channel (mono).

It shows how the frequency content of a sound changes over time using a single combined audio signal, not left/right channels.

3.9.2 Why Mono-Spectrogram Is Used in Voice Recognition?

- ✓ Simpler than stereo.
- ✓ Lower computational cost.
- ✓ Consistent across microphones.
- ✓ Focuses on speaker characteristics, not direction.

3.9.3 Mono-Spectrogram in Voice Recognition Pipeline:

1. Record voice (mono).
2. Preprocess (noise reduction, normalization).
3. Convert to mono (if needed).
4. Generate spectrogram.
5. Extract features (MFCC / CNN embeddings).
6. Compare using cosine similarity.

3.10 Power Spectrum

3.10.1 What is the Power Spectrum?

The power spectrum shows how the power (energy) of a signal is distributed across different frequencies.

In voice and signal processing, it tells us which frequencies contain most of the speech energy.

3.10.2 Power Spectrum in Speech Processing:

3.10.2.1 Human speech typically contains:

- ✓ Fundamental frequency (pitch): ~85–255 Hz
- ✓ Formants: 300–3400 Hz

3.10.2.2 The power spectrum highlights:

- ✓ Pitch energy.
- ✓ Formant structure.
- ✓ Speaker-specific traits.

3.10.3 Power Spectrum in Voice Recognition

The power spectrum is used to:

- ➢ Extract MFCC.
- ➢ Create spectrogram features.
- ➢ Feed CNN models.
- ➢ Analyze speaker characteristics.
- ➢ It is a foundation step, not a final classifier.

3.10.4 Advantages

- ✓ Reveals frequency energy distribution.
- ✓ Essential for speech feature extraction.
- ✓ Works well with statistical & ML methods.

3.10.5 Limitations

- ✓ Sensitive to noise.
- ✓ Loses phase information.
- ✓ Requires preprocessing for robustness.

3.11 Log-Mel Spectrogram

3.11.1 What is it?

A Log-Mel spectrogram is a time–frequency representation of speech where:

- Frequencies are mapped to the Mel scale (human hearing scale).
- Energy is converted to logarithmic scale.

3.11.2 Why Mel Scale?

Human ears:

- Are more sensitive to low frequencies
- Have lower resolution at high frequencies
- The Mel scale mimics this perception.

3.12 Delta MFCC (ΔMFCC)

3.12.1 What are Delta MFCCs?

- Delta MFCCs represents the rate of change of MFCCs over time.
- They capture speech dynamics, not just static information.

3.12.2 Why Delta MFCC is important

1. Speech is dynamic:

- ✓ Same speaker varies pronunciation.
- ✓ Temporal changes matter.

2. Delta MFCC captures:

- ✓ Transitions between phonemes.
- ✓ Speaking style.

3.13 Machine Learning Algorithms

3.13.1 Decision Tree 

3.13.1.1 What it is:

A Decision Tree is a model that makes decisions by asking a series of yes/no questions, like a flowchart.

3.13.1.2 How it works?

- It starts at the root (first question).
- Each question splits the data into branches.
- At the end (leaf), it gives a final decision.

3.13.1.3 Example:

- "Should I open the door?"
- Is the voice recognized? → Yes / No
- Is confidence > 90%? → Yes / No
- ➞ Decision: Open / Don't open

3.13.1.4 Pros:

- ✓ Easy to understand
- ✓ Easy to visualize
- ✓ Fast

3.13.1.5 Cons:

- ✓ Can overfit (memorize data)
- ✓ Not very accurate alone for complex problems

3.13.2 Random Forest 🌲🌲🌲

3.13.2.1 What it is:

A Random Forest is a group of many decision trees working together.

3.13.2.2 How it works?

- Builds many different decision trees.
- Each tree gives a prediction.
- Final result = majority vote (classification) or average (regression).

3.13.2.3 Example:

- 100 trees check the voice.
- 80 say “authorized”, 20 say “not”.

  ➞ Final decision: authorized

3.13.2.4 Pros:

- ✓ More accurate than a single decision tree.
- ✓ Reduces overfitting.
- ✓ Works well on many problems.

3.13.2.5 Cons:

- ✓ Slower than one tree.
- ✓ Harder to interpret.

3.13.3 SVM (Support Vector Machine)

3.13.3.1 What it is

SVM is a model that separates data using a line or boundary (called a hyperplane).

3.13.3.2 How it works

- Finds the best separating line between classes.
- Maximizes the margin (distance) between classes.
- Can use kernels to separate complex data.

3.13.3.3 Example:

- Voice recognition:
- One side → authorized voices
- Other side → unauthorized voices
- The model finds the clearest boundary between them.

3.13.3.4 Pros:

- ✓ Very powerful for small/medium datasets.
- ✓ Works well with high-dimensional data (audio, images).
- ✓ Good accuracy.

3.13.3.5 Cons:

- ✓ Slow on large datasets.
- ✓ Hard to tune (kernel, parameters).

3.13.4 XGBoost 🚀

3.13.4.1 What it is:

XGBoost is an advanced version of decision trees, trained sequentially to fix previous mistakes.

3.13.4.2 How it works?

- Builds trees one by one.
- Each new tree focuses on correcting errors of the previous trees.
- Uses optimization and regularization.

3.13.4.3 Example:

- First tree misclassifies some voices.
- Second tree focuses on those mistakes.
- Third tree improves more.
- ➞ Very accurate final model

3.13.4.1 Pros:

- ✓ Extremely high accuracy.
- ✓ Fast and efficient.
- ✓ Winner in many ML competitions.

3.13.4.1 Cons:

- ✓ Complex.
- ✓ Hard to understand visually.
- ✓ Needs careful tuning.

Feature	Decision Tree	Random Forest	SVM	XGBoost
Type	Single tree	Ensemble of trees	Margin-based model	Boosted trees
Complexity	Very low	Medium	High	Very high
Accuracy	Low–Medium	High	High	Very High
Overfitting	High	Low	Low	Very Low
Speed (Training)	Very fast	Medium	Slow (large data)	Medium–Fast
Speed (Prediction)	Very fast	Medium	Fast	Fast
Interpretability	Very easy	Hard	Hard	Very hard
Handles Non-linearity	Limited	Good	Very good (kernels)	Excellent
Works with Large Data	✗	✓	✗	✓
Feature Importance	✓	✓	✗	✓

Table 7-3: Comparison table of machine language algorithm.

❖Chapter 4: Results and Analysis

4.1.1 The principle of Code Operation

The code's function is to implement what was explained previously in the report regarding the topic of feature extraction. The code records 3 attempts of the speaker, then purifies them as much as possible from noise by raising the sound Amplitude for high-value sounds, and reducing the Amplitude of the sounds with low values. Then it performs extraction operations for the mentioned values (MFCC, PP, FF), prints their values, and draws curves representing them using ready-made libraries.

4.1.2 Code Result for an 11-year-old child

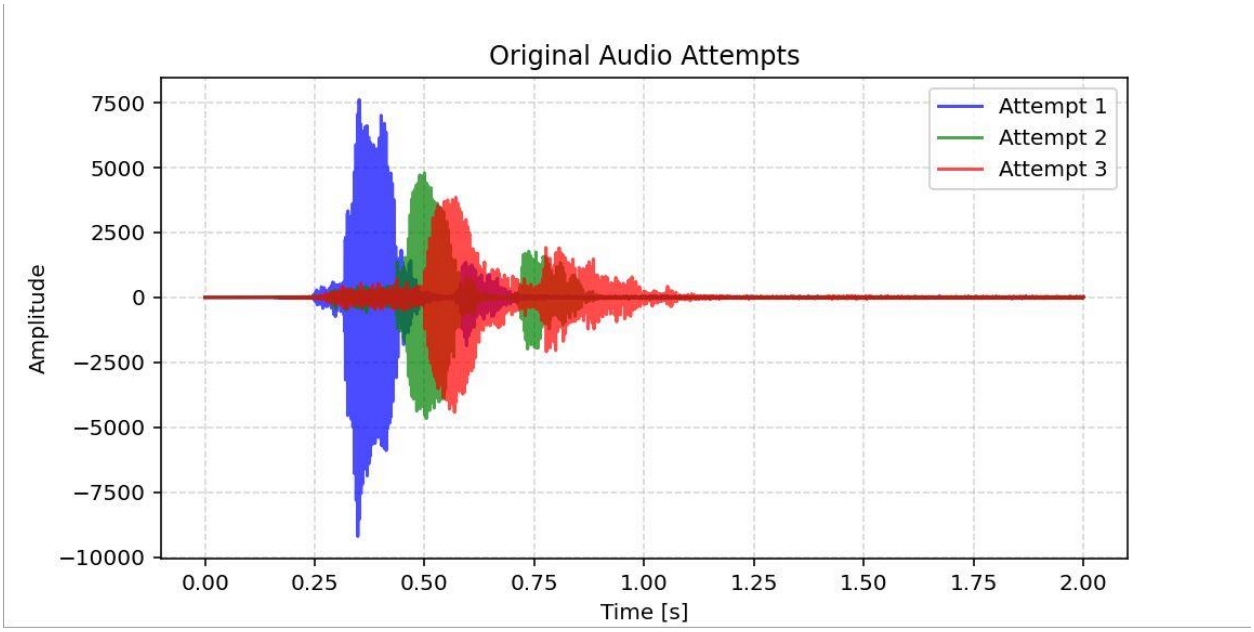


Figure 2-4: The child's Original Audio in the three Attempts.

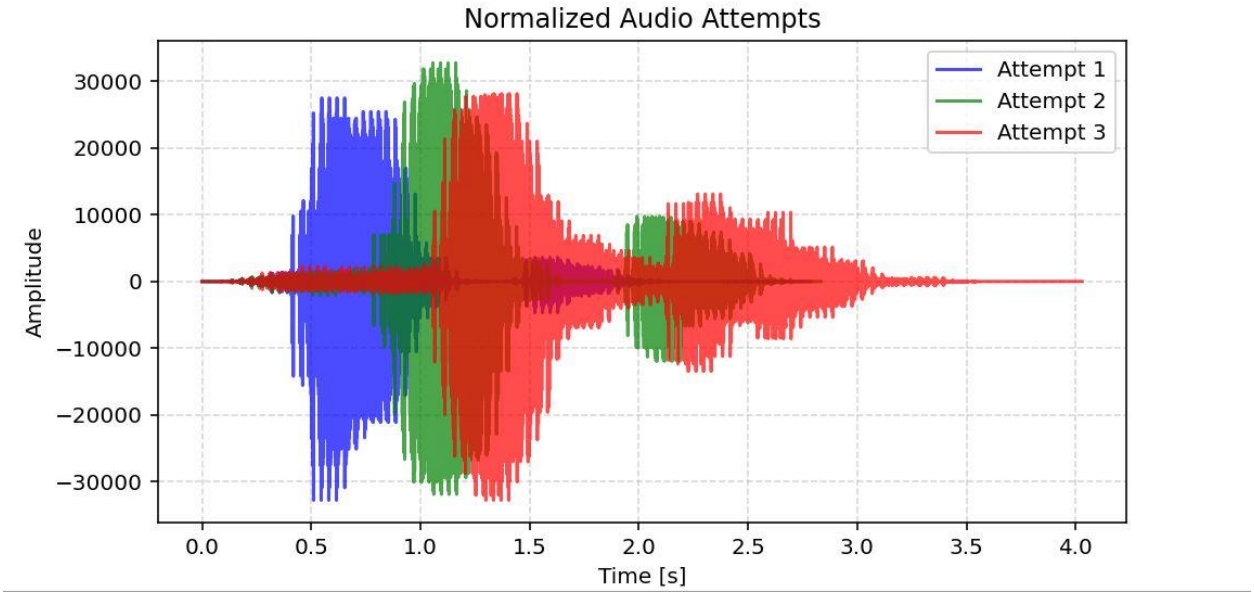


Figure 3-4: The child's Normalized Audio in the three Attempts.

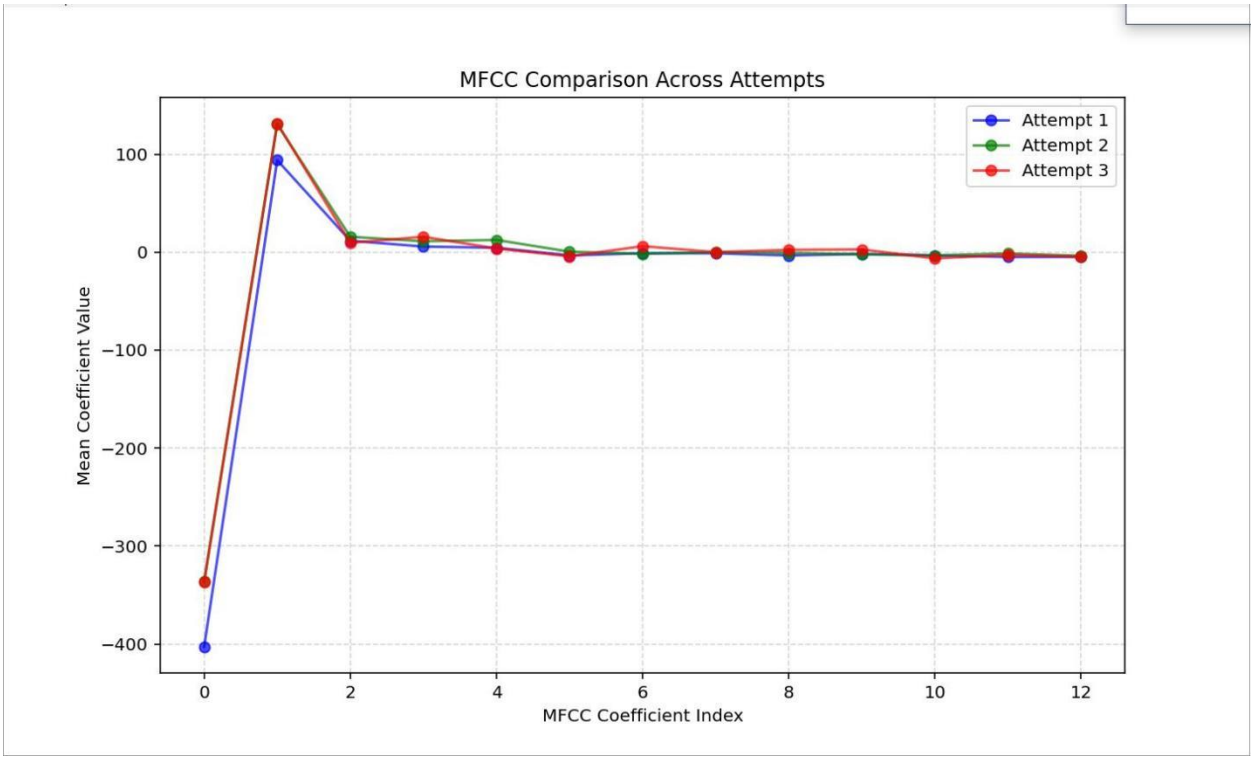
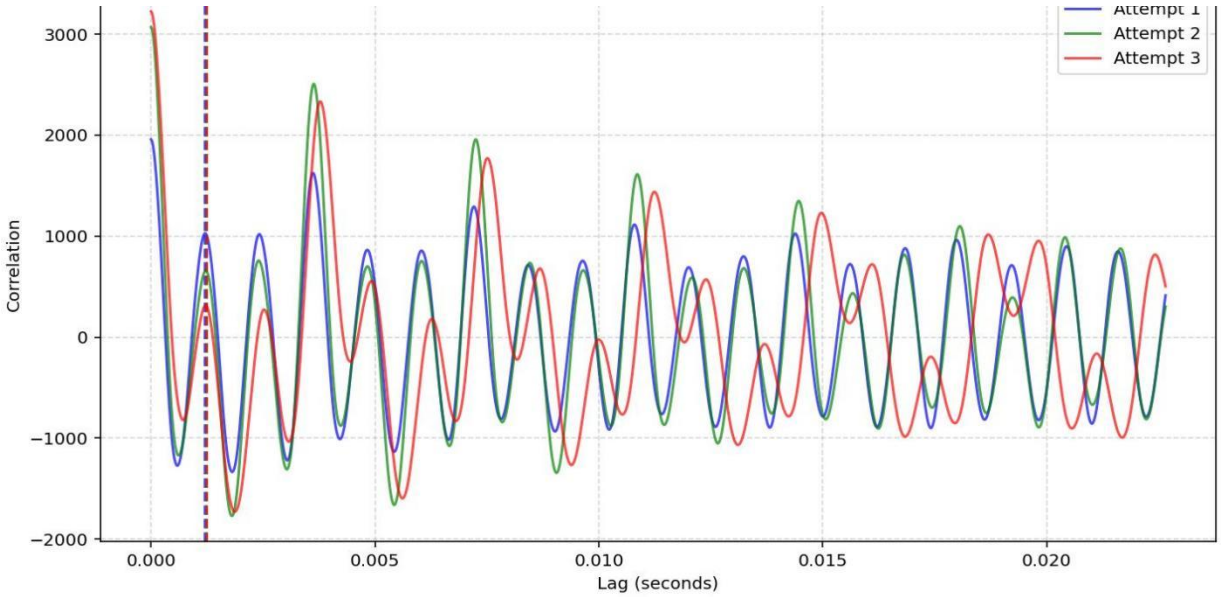


Figure 4-4: The child's MFCC comparison across in the three Attempts.



Pitch Analysis Results:

Attempt 1:
• Pitch Period: 0.0012 s
• Fundamental Freq: 832.08 Hz

Attempt 2:
• Pitch Period: 0.0012 s
• Fundamental Freq: 816.67 Hz

Attempt 3:
• Pitch Period: 0.0012 s
• Fundamental Freq: 816.67 Hz

Figure 5-4: The child's Pitch Period and Fundamental Frequency value in the three Attempts.

Time (s)
Formant Analysis Results:

Attempt 1 Median Formants:
F1: 503.9 Hz
F2: 1430.8 Hz
F3: 2247.2 Hz
F4: 3235.4 Hz

Attempt 2 Median Formants:
F1: 472.9 Hz
F2: 1399.1 Hz
F3: 2238.4 Hz
F4: 3167.8 Hz

Attempt 3 Median Formants:
F1: 502.5 Hz
F2: 1262.9 Hz
F3: 2170.9 Hz
F4: 2998.0 Hz

Figure 6-4: The child's Fomant Analysis Result in the three Attempts.

4.1.3 Code Result for Female

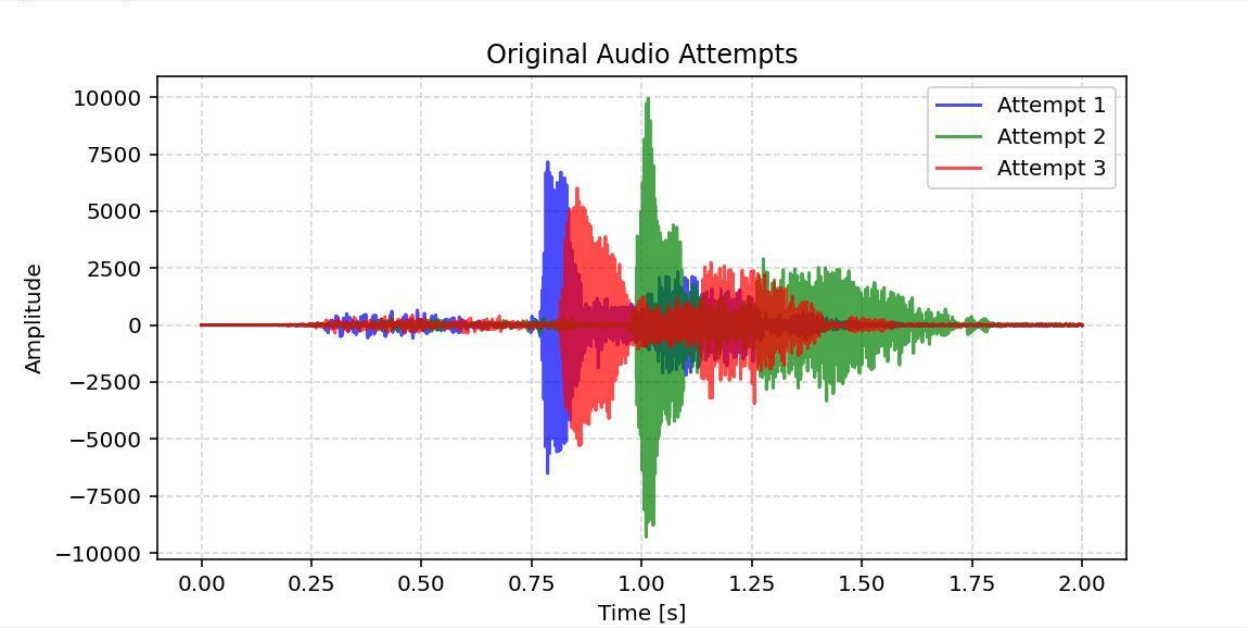


Figure 7-4: Female Original Audio in the three Attempts.

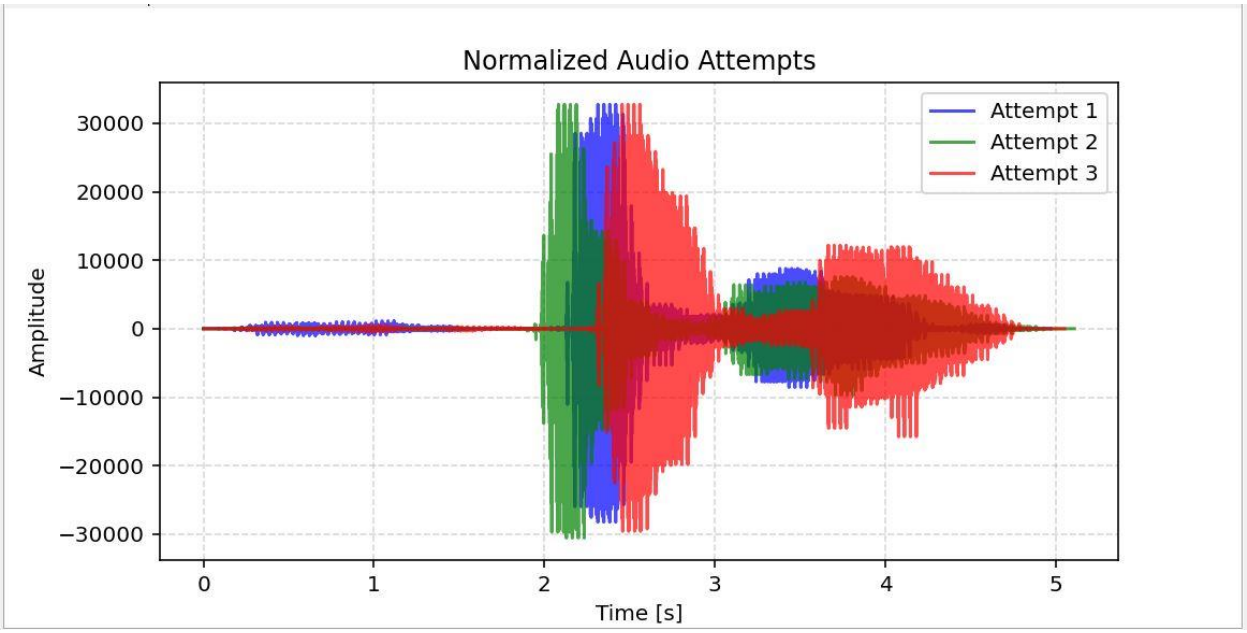


Figure 8-4: Female Normalized Audio in the three Attempts.

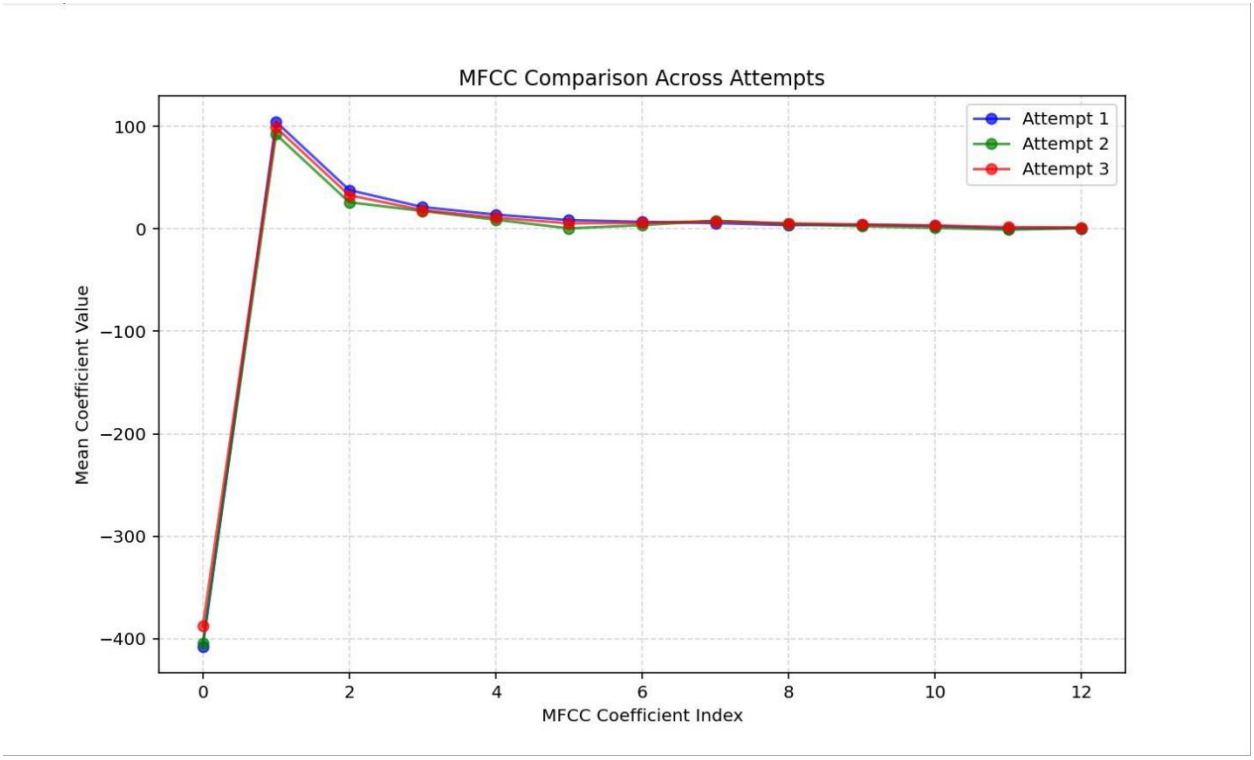


Figure 9-4: Female MFCC comparison across in the three Attempts.

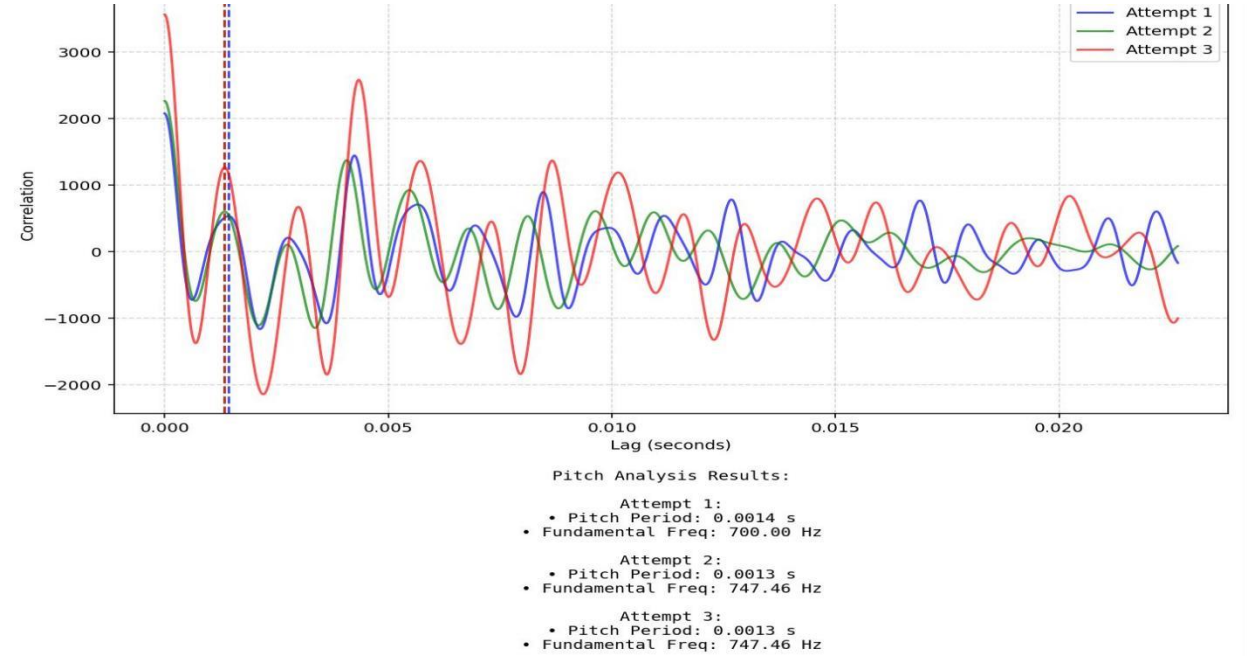


Figure 10-4:* Female *pitch Period and Fundamental Frequency value in the three Attempts.

```
Time (s)
Formant Analysis Results:

Attempt 1 Median Formants:
        F1: 266.0 Hz
        F2: 1341.1 Hz
        F3: 2243.8 Hz
        F4: 3084.0 Hz

Attempt 2 Median Formants:
        F1: 300.9 Hz
        F2: 1298.7 Hz
        F3: 2247.7 Hz
        F4: 3122.5 Hz

Attempt 3 Median Formants:
        F1: 275.1 Hz
        F2: 1366.0 Hz
        F3: 2220.2 Hz
        F4: 3088.7 Hz
```

Figure 11-4: **Female Formant Analysis Results** *in the three* **Attempts.**

4.1.4 Code Result for Male

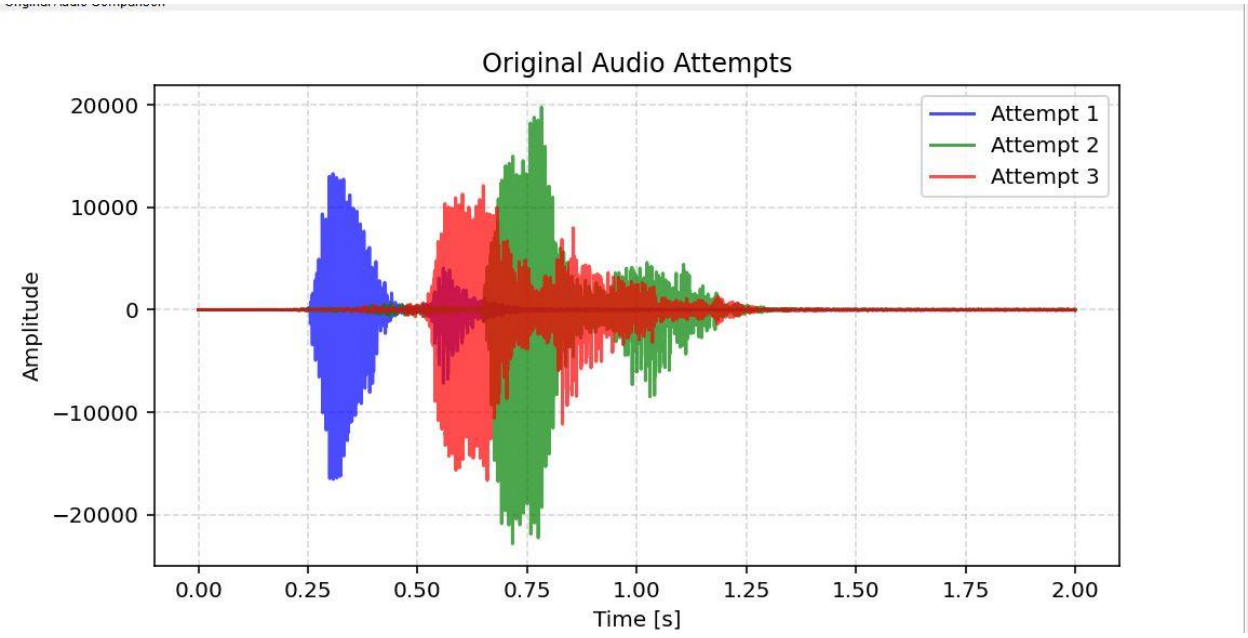


Figure 12-4: **Male** *Original* **Audio** *in the three* **Attempts.**

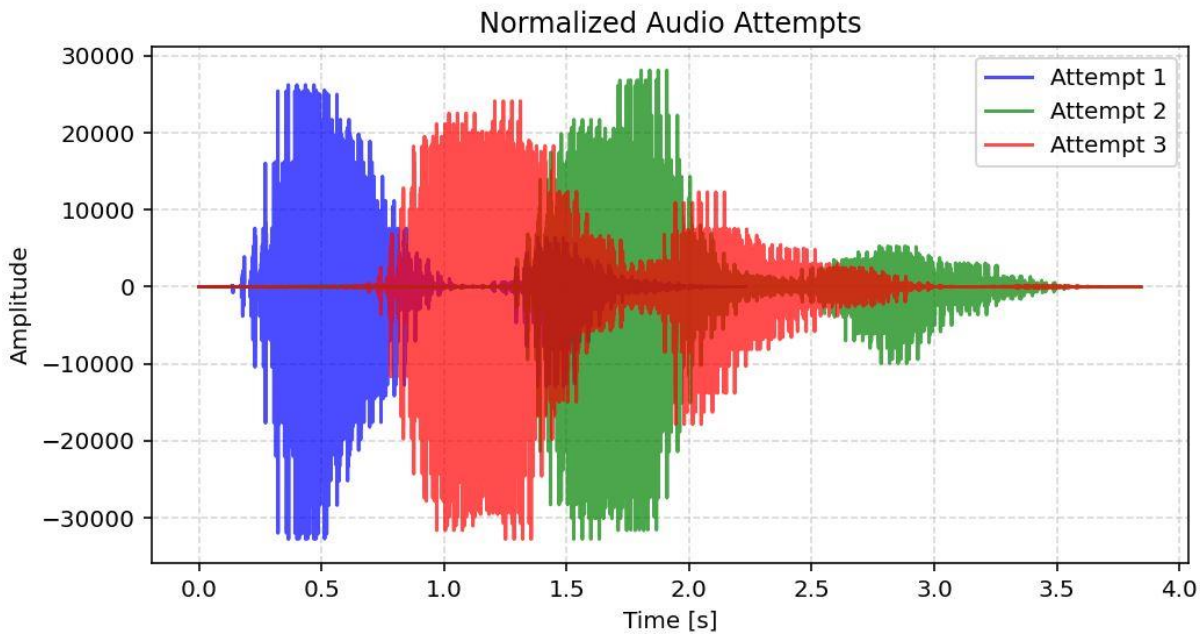


Figure 13-4: **Male Normalized Audio *in the three* Attempts.**

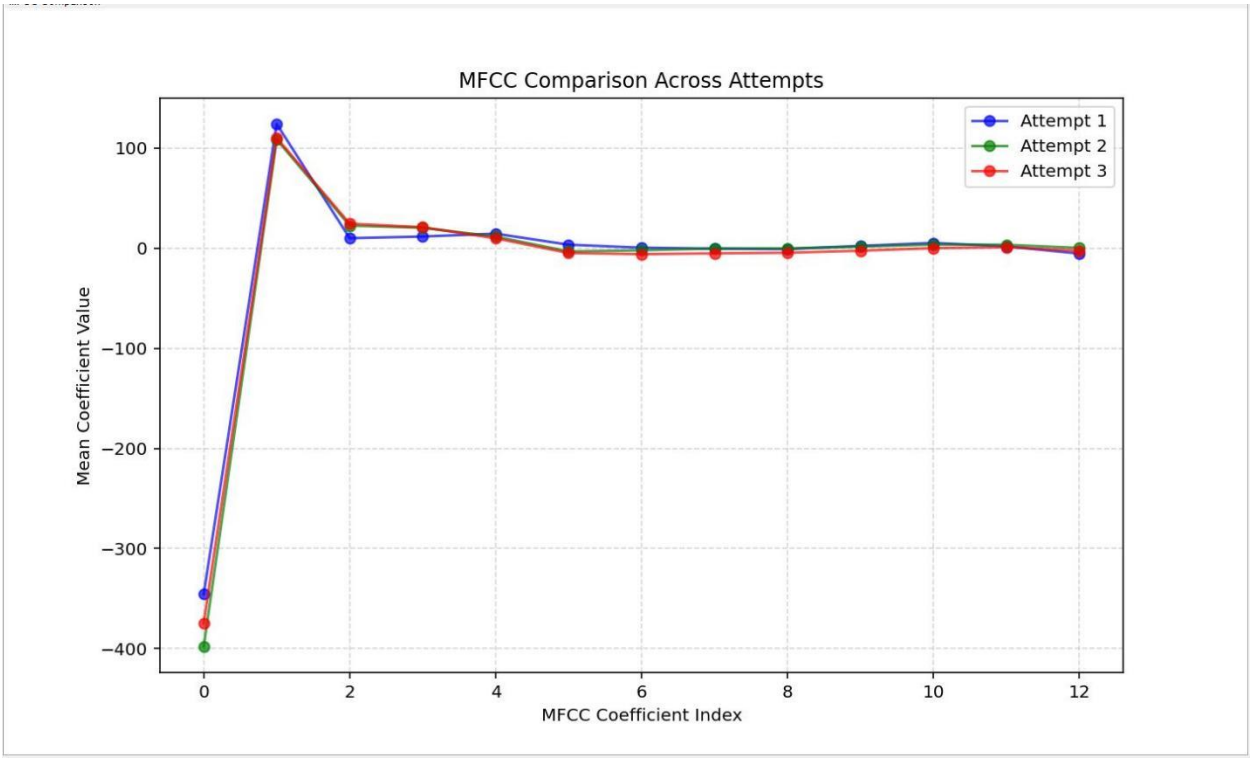


Figure 14-4: **Male MFCC comparison across *in the three* Attempts.**

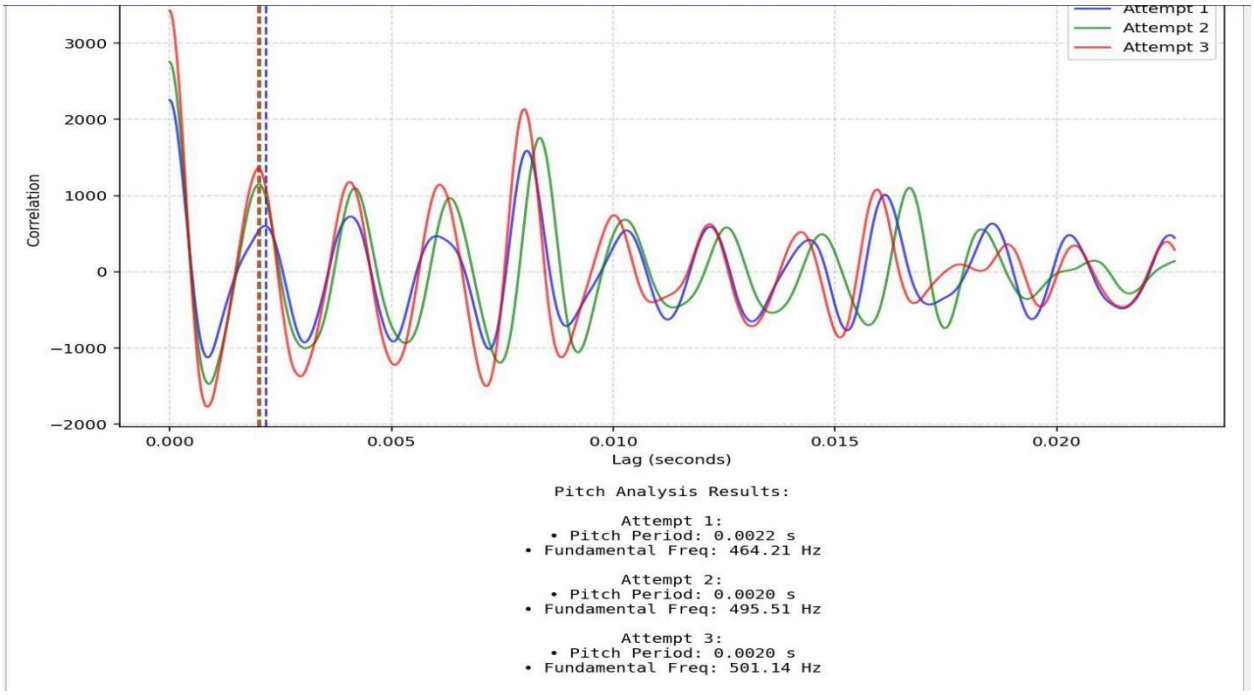


Figure 15-4: **Male** ***pitch Period and Fundamental Frequency value in the three Attempts.***

```
Time (s)
Formant Analysis Results:

Attempt 1 Median Formants:
        F1:  504.3 Hz
        F2: 1524.8 Hz
        F3: 2247.1 Hz
        F4: 3280.2 Hz

Attempt 2 Median Formants:
        F1:  540.8 Hz
        F2: 1404.6 Hz
        F3: 2492.9 Hz
        F4: 3409.3 Hz

Attempt 3 Median Formants:
        F1:  496.8 Hz
        F2: 1309.6 Hz
        F3: 2352.7 Hz
        F4: 3386.9 Hz
```

Figure 16-4: **Male Formant Analysis Results** ***in the three Attempts.***

4.2.1 Pre-Train Code

```
import numpy as np
import scipy
import librosa
import soundfile as sf
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import cosine_similarity
import os
import torch
import torchaudio
from speechbrain.inference.speaker import SpeakerRecognition



from pathlib import Path


from speechbrain.utils.fetching import LocalStrategy


TARGET_SR = 16000
N_MELS = 256
TARGET_FRAMES = 256
```

```
# ---------- Your functions (same logic) ----------
def wav_to_mel256(wav_path, target_sr=TARGET_SR, n_mels=N_MELS,
target_frames=TARGET_FRAMES):
    x, sr = sf.read(wav_path)
    if x.ndim == 2:
        x = x.mean(axis=1)
    if sr != target_sr:
        x = librosa.resample(x.astype(np.float32), orig_sr=sr, target_sr=target_sr)
        sr = target_sr
    else:
        x = x.astype(np.float32)
    S = librosa.feature.melspectrogram(
        y=x, sr=sr,
        n_fft=1024,
        hop_length=256,
        n_mels=n_mels,
        power=2.0
    )
    S_db = librosa.power_to_db(S, ref=np.max)
    frames = S_db.shape[1]
    if frames < target_frames:
        pad = target_frames - frames
        S_db = np.pad(S_db, ((0, 0), (0, pad)), mode="constant", constant_values=S_db.min())
    else:
        S_db = S_db[:, :target_frames]

    S_db = (S_db - S_db.min()) / (S_db.max() - S_db.min() + 1e-9)
    return S_db.astype(np.float32)
```

```
def save_spectrogram(spec256, save_dir, base_name):
    os.makedirs(save_dir, exist_ok=True)
    npy_path = os.path.join(save_dir, base_name + ".npy")
    png_path = os.path.join(save_dir, base_name + ".png")

    np.save(npy_path, spec256)

    plt.figure(figsize=(4, 4))
    plt.imshow(spec256, origin="lower", aspect="auto", cmap="magma")
    plt.axis("off")
    plt.tight_layout(pad=0)
    plt.savefig(png_path, dpi=300, bbox_inches="tight", pad_inches=0)
    plt.close()

    return npy_path, png_path

def pitch_period_from_spectrogram_cepstrum(
    wav_path, target_sr=TARGET_SR, n_fft=1024, hop_length=256, fmin=50, fmax=400
):
    x, sr = sf.read(wav_path)
    if x.ndim == 2:
        x = x.mean(axis=1)
    x = x.astype(np.float32)

    if sr != target_sr:
        x = librosa.resample(x, orig_sr=sr, target_sr=target_sr)
        sr = target_sr
```

```
def pitch_period_from_spectrogram_cepstrum(
    wav_path, target_sr=TARGET_SR, n_fft=1024, hop_length=256, fmin=50, fmax=400
):
    x, sr = sf.read(wav_path)
    if x.ndim == 2:
        x = x.mean(axis=1)
    x = x.astype(np.float32)
    if sr != target_sr:
        x = librosa.resample(x, orig_sr=sr, target_sr=target_sr)
        sr = target_sr
    S = np.abs(librosa.stft(x, n_fft=n_fft, hop_length=hop_length, window="hann")) + 1e-12
    logS = np.log(S)
    C = np.fft.irfft(logS, axis=0)
    qmin = int(sr / fmax)
    qmax = int(sr / fmin)
    qmax = min(qmax, C.shape[0]-1)
    periods = np.zeros(C.shape[1], dtype=np.float32)
    for t in range(C.shape[1]):
        seg = C[qmin:qmax, t]
        k = np.argmax(seg) + qmin
        periods[t] = k / sr
    median_period = float(np.median(periods))
    median_f0 = 1.0 / median_period if median_period > 0 else 0.0
    return periods, median_period, median_f0
```

```
# ---------- SpeechBrain model (load once) ----------
_spkrec = None
def get_spkrec(device=None, savedir=None):
    global _spkrec
    if _spkrec is not None:
        return _spkrec
    if device is None:
        device = "cuda" if torch.cuda.is_available() else "cpu"
    if savedir is None:
        savedir = os.path.join(os.path.expanduser("~"), "sb_models", "pretrained_ecapa")
    os.makedirs(savedir, exist_ok=True)
    _spkrec = SpeakerRecognition.from_hparams(
        source="speechbrain/spkrec-ecapa-voxceleb",
        savedir=savedir,
        local_strategy=LocalStrategy.COPY,
        run_opts={"device": device},)
    return _spkrec
def extract_ecapa_embedding(wav_path, device=None):
    spkrec = get_spkrec(device=device)
    signal, sr = torchaudio.load(wav_path)
    if signal.shape[0] > 1:
        signal = signal.mean(dim=0, keepdim=True)
    if sr != TARGET_SR:
        signal = torchaudio.transforms.Resample(sr, TARGET_SR)(signal)
    with torch.no_grad():
        emb =
spkrec.encode_batch(signal.to(spkrec.device)).squeeze().detach().cpu().numpy().astype(np.float32)
    emb = emb / (np.linalg.norm(emb) + 1e-9)
    return emb
```

```
def extract_all_features(wav_path, save_dir):
    """
    Main function the GUI will call.
    Returns dict with paths + pitch + embedding
    """
    wav_path = str(wav_path)
    save_dir = str(save_dir)
    base = Path(wav_path).stem  # safe base name (no slashes)
    # 1) spectrogram
    spec = wav_to_mel256(wav_path)
    spec_npy, spec_png = save_spectrogram(spec, save_dir, base)
    # 2) pitch
    _, T0_med, f0_med = pitch_period_from_spectrogram_cepstrum(wav_path)
    # 3) embedding
    emb = extract_ecapa_embedding(wav_path)
    emb_path = os.path.join(save_dir, base + "_ecapa192.npy")
    np.save(emb_path, emb)
    return {
        "spec_shape": tuple(spec.shape),
        "spec_npy": spec_npy,
        "spec_png": spec_png,
        "T0_med": float(T0_med),
        "F0_med": float(f0_med),
        "emb_path": emb_path,
        "emb_dim": int(emb.shape[0]),
    }
```

```
wav_path = r"D:\anaconda_app\my_works\twins_samples\tima\wav_samples/1.wav"  # غيّر المسار
# تحميل الصوت
signal, sr = sf.read(wav_path)
print("Sample rate:", sr)
print("Signal shape:", signal.shape)
print("Data type:", signal.dtype)
```

```
def wav_to_mel256(wav_path, target_sr=16000, n_mels=256, target_frames=256):
    # 1) Load wav
    x, sr = sf.read(wav_path)
    # 2) Convert to mono if stereo
    if x.ndim == 2:
        x = x.mean(axis=1)
    # 3) Resample to fixed SR (important to make shapes consistent)
    if sr != target_sr:
        x = librosa.resample(x.astype(np.float32), orig_sr=sr, target_sr=target_sr)
        sr = target_sr
    else:
        x = x.astype(np.float32)
    # 4) Mel Spectrogram (freq axis fixed = 256)
    S = librosa.feature.melspectrogram(
        y=x, sr=sr,
        n_fft=1024,
        hop_length=256,
        n_mels=n_mels,
        power=2.0)
    # 5) Convert to dB (log scale)
    S_db = librosa.power_to_db(S, ref=np.max)
    # 6) Fix time axis to 256 frames (pad or crop)
    frames = S_db.shape[1]
    if frames < target_frames:
        pad = target_frames - frames
        S_db = np.pad(S_db, ((0, 0), (0, pad)), mode="constant", constant_values=S_db.min())
    else:
        S_db = S_db[:, :target_frames]
    # 7) Normalize to [0, 1] (nice for CNN)
    S_db = (S_db - S_db.min()) / (S_db.max() - S_db.min() + 1e-9)
    # Final shape: (256, 256)
    return S_db.astype(np.float32)       spec256 = wav_to_mel256(wav_path)
```

```
def save_spectrogram(spec256, save_dir, base_name):
    """
    spec256 : numpy array (256, 256)
    save_dir: folder path
    base_name: file name without extension
    """
    # 1) تأكد أن المجلد موجود
    os.makedirs(save_dir, exist_ok=True)
    # 2) مسارات الحفظ
    npy_path = os.path.join(save_dir, base_name + ".npy")
    png_path = os.path.join(save_dir, base_name + ".png")
    # 3) حفظ npy (القيم الأصلية)
    np.save(npy_path, spec256)
    # 4) حفظ png (كصورة)
    plt.figure(figsize=(4, 4))
    plt.imshow(spec256, origin="lower", aspect="auto", cmap="magma")
    plt.axis("off")
    plt.tight_layout(pad=0)
    plt.savefig(png_path, dpi=300, bbox_inches="tight", pad_inches=0)
    plt.close()
    print("Saved:")
    print(npy_path)
    print(png_path)
# ===== Example =====
save_folder = r"D:/anaconda_app/my_works"   # اختر المسار
file_name = "D:\anaconda_app\my_works\twins_samples\tima\wav_samples/1.wav"
save_spectrogram(spec256, save_folder, file_name)
```

```
def pitch_period_from_spectrogram_cepstrum(
    wav_path,
    target_sr=16000,
    n_fft=1024,
    hop_length=256,
    fmin=50,
    fmax=400
):
    # 1) load wav
    x, sr = sf.read(wav_path)
    if x.ndim == 2:
        x = x.mean(axis=1)
    x = x.astype(np.float32)
    # 2) resample (for stable ranges)
    if sr != target_sr:
        x = librosa.resample(x, orig_sr=sr, target_sr=target_sr)
        sr = target_sr
    # 3) STFT magnitude spectrogram
    S = np.abs(librosa.stft(x, n_fft=n_fft, hop_length=hop_length, window="hann")) + 1e-12
    logS = np.log(S)
    # 4) cepstrum per frame (IFFT over frequency axis)
    C = np.fft.irfft(logS, axis=0)  # shape: (quefrency_bins, frames)
    # 5) search range in quefrency (seconds -> samples)
    qmin = int(sr / fmax)   # period samples for fmax
    qmax = int(sr / fmin)   # period samples for fmin
    qmax = min(qmax, C.shape[0]-1)
```

```
  # 6) pick peak -> pitch period per frame
    periods = np.zeros(C.shape[1], dtype=np.float32)
    for t in range(C.shape[1]):
        seg = C[qmin:qmax, t]
        k = np.argmax(seg) + qmin
        periods[t] = k / sr  # seconds

    # optional: return median period (more stable)
    median_period = float(np.median(periods))
    median_f0 = 1.0 / median_period if median_period > 0 else 0.0

    return periods, median_period, median_f0



# ===== Example =====
#wav_path = r"D:/anaconda app/my works/my sample/clean samples/WhatsApp Audio 2025-11-05 at 23.33.15_0487ced2.dat.wav"  # غيّر المسار
periods, T0_med, f0_med = pitch_period_from_spectrogram_cepstrum(wav_path)

print("Median pitch period (sec):", T0_med)
print("Median F0 (Hz):", f0_med)
print("Frames:", len(periods))
```

```
def extract_pitch_period_wav(
    wav_path,
    target_sr=16000,
    n_fft=1024,
    hop_length=256,
    fmin=50,
    fmax=400,
    energy_percentile=65,   # إذا الضوضاء عالية (70) ارفعها
    min_voiced_ms=300       # أقل طول مقطع voiced نعتبره صالح):
    # 1) Load WAV
    x, sr = sf.read(wav_path)
    if x.ndim == 2:
        x = x.mean(axis=1)
    x = x.astype(np.float32)
    # 2) Resample (يثبت الحدود fmin/fmax)
    if sr != target_sr:
        x = librosa.resample(x, orig_sr=sr, target_sr=target_sr)
        sr = target_sr
    # 3) STFT magnitude (للـ cepstrum)
    S = np.abs(librosa.stft(x, n_fft=n_fft, hop_length=hop_length, window="hann")) + 1e-12
    logS = np.log(S)
    # 4) Voiced mask من طاقة الإطار (قص الإشارة/بدون حذف)
    frame_energy = S.mean(axis=0)  # (frames,)
    thr = np.percentile(frame_energy, energy_percentile)
    voiced = frame_energy >= thr
    if not np.any(voiced):
        return None, None, 0.0, None  # no voiced frames
```

```
# 5) اختيار أطول مقطع voiced متصل (أكثر ثبات)
  idx = np.where(voiced)[0]
  runs = []
  start = idx[0]
  prev = idx[0]
  for k in idx[1:]:
    if k == prev + 1:
      prev = k
    else:
      runs.append((start, prev))
      start = prev = k
  runs.append((start, prev))
  best_start, best_end = max(runs, key=lambda ab: ab[1] - ab[0])
  # تحقق من طول المقطع
  frames_len = best_end - best_start + 1
  seg_ms = (frames_len * hop_length / sr) * 1000.0
  if seg_ms < min_voiced_ms:
    # fallback: استخدم كل الإطارات voiced (تلصق الإشارة/بدون ما تقطع)
    best_start, best_end = idx[0], idx[-1]
# 6) Cepstrum على الإطارات داخل المقطع فقط
C = np.fft.irfft(logS, axis=0)  # (quef_bins, frames)
qmin = int(sr / fmax)     # أصغر فترة (عينات) لف0 الأعلى
qmax = int(sr / fmin)     # أكبر فترة (عينات) لف0 الأدنى
qmax = min(qmax, C.shape[0] - 1)
T0_list = []
for t in range(best_start, best_end + 1):
  seg = C[qmin:qmax, t]
  k = int(np.argmax(seg)) + qmin
  T0_list.append(k / sr)  # seconds
T0_per_frame = np.array(T0_list, dtype=np.float32)
T0_med = float(np.median(T0_per_frame))
F0_med = 1.0 / T0_med if T0_med > 0 else 0.0
# نرجّع كمان حدود المقطع (frames) إذا احتجتها لاحقًا
segment_info = {"start_frame": int(best_start), "end_frame": int(best_end), "sr": int(sr)}
return T0_per_frame, T0_med, float(F0_med), segment_info
```

```
device = "cuda" if torch.cuda.is_available() else "cpu"
print("Device:", device)
# 2) حمّل نموذج ECAPA pretrained
from speechbrain.utils.fetching import LocalStrategy  # <-- important
savedir = os.path.join(os.path.expanduser("~"), "sb_models", "pretrained_ecapa")
os.makedirs(savedir, exist_ok=True)
spkrec = SpeakerRecognition.from_hparams(
   source="speechbrain/spkrec-ecapa-voxceleb",
   savedir=savedir,
   local_strategy=LocalStrategy.COPY  # <-- no symlink, fixes WinError 1314)
# 3) اقرأ wav (يفضل mono)
wav_path = r"D:\anaconda_app\my_works\twins_samples\tima\wav_samples/1.wav"  # <-- غيّر المسار
signal, sr = torchaudio.load(wav_path)  # signal: (channels, samples)
# 4) حوّل إلى mono لو stereo
if signal.shape[0] > 1:
   signal = signal.mean(dim=0, keepdim=True)
# 5) resample إلى 16kHz (هذا مهم للنموذج)
target_sr = 16000
if sr != target_sr:
   resampler = torchaudio.transforms.Resample(sr, target_sr)
   signal = resampler(signal)
   sr = target_sr
# 6) استخراج embedding
with torch.no_grad():
   emb = spkrec.encode_batch(signal.to(device))  # shape غالباً: (1, 1, D) أو (1, D)
   emb = emb.squeeze().detach().cpu().numpy().astype(np.float32)
print("Embedding shape:", emb.shape)
# 7) (اختياري) Normalize embedding (مفيد للكوساين)
emb = emb / (np.linalg.norm(emb) + 1e-9)
# 8) (اختياري) حفظ embedding
out_path = r"D:/anaconda_app/my_works/embedding4.npy"
np.save(out_path, emb)
```

4.3.1 Bulit Code

```
import os
import csv
from pathlib import Path


import numpy as np
import torch
import torchaudio


from speechbrain.inference.speaker import SpeakerRecognition
from speechbrain.utils.fetching import LocalStrategy


# ====================== CONFIG ======================
INPUT_DIR  = Path(r"D:\anaconda_app\my_works\twins_samples\tima\wav_samples")
OUTPUT_DIR = Path(r"D:\anaconda_app\my_works\embeddings_out_tima")
MODEL_DIR  = Path(os.path.expanduser(r"~\sb_models\pretrained_ecapa"))
TARGET_SR  = 16000
# ====================================================



def load_model(device: str):
    MODEL_DIR.mkdir(parents=True, exist_ok=True)
    spkrec = SpeakerRecognition.from_hparams(
        source="speechbrain/spkrec-ecapa-voxceleb",
        savedir=str(MODEL_DIR),
        local_strategy=LocalStrategy.COPY,
        run_opts={"device": device},   # ensures model + pipeline use same device
    )
    return spkrec
```

```
def wav_to_16k_mono(wav_path: Path, target_sr: int = 16000) -> torch.Tensor:
    sig, sr = torchaudio.load(str(wav_path))  # (C, T)
    # mono
    if sig.shape[0] > 1:
        sig = sig.mean(dim=0, keepdim=True)
    sig = sig.to(torch.float32)
    # resample
    if sr != target_sr:
        sig = torchaudio.transforms.Resample(sr, target_sr)(sig)
    return sig  # still on CPU here (fine)
def extract_embedding(spkrec, sig: torch.Tensor) -> np.ndarray:
    # move input to the same device SpeechBrain is using (cuda/cpu)
    sig = sig.to(spkrec.device)
    with torch.no_grad():
        emb = spkrec.encode_batch(sig).squeeze().detach().cpu().numpy().astype(np.float32)
    # L2 normalize for cosine similarity
    emb /= (np.linalg.norm(emb) + 1e-9)
    return emb
def main():
    OUTPUT_DIR.mkdir(parents=True, exist_ok=True)
    device = "cuda" if torch.cuda.is_available() else "cpu"
    print("Device:", device)
    spkrec = load_model(device)
    wav_files = sorted([p for p in INPUT_DIR.rglob("*") if p.suffix.lower() == ".wav"])
    print("Found WAV files:", len(wav_files))


    index_path = OUTPUT_DIR / "index.csv"
    ok_count, fail_count = 0, 0
```

```
import numpy as np
from pathlib import Path
# ====== CONFIG ======
DATASET_DIR = Path(r"D:/anaconda_app/my_works/embeddings_out2")    # embeddings dataset folder
TEST_EMB    = Path(r"D:/anaconda_app/my_works/embedding4.npy")     # <-- test embedding from ANY folder
THRESHOLD   = 0.72
TOPK        = 10
# ====================
def load_emb(p: Path) -> np.ndarray:
    e = np.load(p).astype(np.float32).reshape(-1)
    e = e / (np.linalg.norm(e) + 1e-9)
    return e
def cosine(a: np.ndarray, b: np.ndarray) -> float:
    return float(np.dot(a, b))  # because both are normalized
# 1) Load test embedding
test = load_emb(TEST_EMB)
# 2) Compare against dataset embeddings
scores = []
for p in DATASET_DIR.glob("*_ecapa192.npy"):
    try:
        e = load_emb(p)
        s = cosine(test, e)
        scores.append((p, s))
    except Exception as ex:
        print("[SKIP]", p.name, "->", ex)
scores.sort(key=lambda x: x[1], reverse=True)
print("Top matches:")
for p, s in scores[:TOPK]:
    print(f"{p.name:50s}  {s:.4f}")
best_p, best_score = scores[0]
print("\nBest match:", best_p.name, "score:", best_score)
print("Decision:", "SAME speaker" if best_score >= THRESHOLD else "DIFFERENT speaker")
print("Threshold:", THRESHOLD)
```

```
import os


BASE_DIR =  r"D:/anaconda_app/my_works/voices/voices"
for d in os.listdir(BASE_DIR):
    print(d, "→", len(os.listdir(os.path.join(BASE_DIR, d))), "wav files")


import os, glob


expected = ["Others_open_100", "Noor_open_85", "Noor_open_20", "Noor_diff_99"]


print("BASE_DIR:", BASE_DIR)
print("Folders found:", os.listdir(BASE_DIR))


for d in expected:
    p = os.path.join(BASE_DIR, d)
    wavs = glob.glob(os.path.join(p, "*.wav"))
    print(f"{d:16s} -> {len(wavs)} wav files")


import numpy as np
import pandas as pd
import librosa
import joblib
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
```

```
FOLDERS = {
   "Others_open_100": 0,
   "Noor_diff_99": 0,
   "Noor_open_85": 1,
   "Noor_open_20": 1,
}


# Speech-friendly params
TARGET_SR = 16000
DURATION_SEC = 2.5


N_FFT = 512
HOP_LENGTH = 160
WIN_LENGTH = 400
N_MELS = 64
N_MFCC = 20


def load_and_fix_length(wav_path, target_sr=TARGET_SR, duration_sec=DURATION_SEC):
   y, sr = librosa.load(wav_path, sr=target_sr, mono=True)
   target_len = int(target_sr * duration_sec)


   # trim/pad
   if len(y) > target_len:
      y = y[:target_len]
   elif len(y) < target_len:
      y = np.pad(y, (0, target_len - len(y)))


   # normalize
   peak = np.max(np.abs(y)) + 1e-9
   y = y / peak
   return y, target_sr
```

```
def extract_features(y, sr):
    feats = []
    # MFCC + delta
    mfcc = librosa.feature.mfcc(
        y=y, sr=sr, n_mfcc=N_MFCC,
        n_fft=N_FFT, hop_length=HOP_LENGTH, win_length=WIN_LENGTH
    )
    mfcc_delta = librosa.feature.delta(mfcc)
    feats.append(np.mean(mfcc, axis=1))
    feats.append(np.std(mfcc, axis=1))
    feats.append(np.mean(mfcc_delta, axis=1))
    feats.append(np.std(mfcc_delta, axis=1))
    # log-mel spectrogram stats
    mel = librosa.feature.melspectrogram(
        y=y, sr=sr, n_fft=N_FFT,
        hop_length=HOP_LENGTH, win_length=WIN_LENGTH,
        n_mels=N_MELS, power=2.0
    )
    logmel = librosa.power_to_db(mel + 1e-9)
    feats.append(np.mean(logmel, axis=1))
    feats.append(np.std(logmel, axis=1))
    # spectral descriptors (mean/std)
    centroid = librosa.feature.spectral_centroid(y=y, sr=sr, n_fft=N_FFT, hop_length=HOP_LENGTH)
    rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr, n_fft=N_FFT, hop_length=HOP_LENGTH)
    bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr, n_fft=N_FFT, hop_length=HOP_LENGTH)
    flatness = librosa.feature.spectral_flatness(y=y, n_fft=N_FFT, hop_length=HOP_LENGTH)
    zcr = librosa.feature.zero_crossing_rate(y, frame_length=WIN_LENGTH, hop_length=HOP_LENGTH)
    for f in [centroid, rolloff, bandwidth, flatness, zcr]:
        feats.append(np.array([np.mean(f), np.std(f)]))
    return np.concatenate(feats).astype(np.float32)
```

```
def build_dataset(base_dir=BASE_DIR):
    rows = []
    for folder_name, label in FOLDERS.items():
        folder_path = os.path.join(base_dir, folder_name)
        wavs = sorted(glob.glob(os.path.join(folder_path, "*.wav")))
        if not wavs:
            print(f"[WARN] No WAV files found in: {folder_path}")
            continue
        for wav_path in wavs:
            try:
                y_audio, sr = load_and_fix_length(wav_path)
                x = extract_features(y_audio, sr)
                rows.append((x, label, folder_name, os.path.basename(wav_path)))
            except Exception as e:
                print(f"[SKIP] {wav_path} -> {e}")
    if not rows:
        raise RuntimeError("No audio loaded. Check BASE_DIR and folder names.")
    X = np.stack([r[0] for r in rows], axis=0)
    y = np.array([r[1] for r in rows], dtype=np.int64)
    meta = pd.DataFrame({
        "folder": [r[2] for r in rows],
        "file": [r[3] for r in rows],
        "label": y
    })
    return X, y, meta
X, y, meta = build_dataset(BASE_DIR)
print("Total samples:", len(y))
print("Class balance:", {0: int((y==0).sum()), 1: int((y==1).sum())})
meta
```

```
def get_models_and_grids():
    models = {}
    # Decision Tree
    dt = DecisionTreeClassifier(random_state=42, class_weight="balanced")
    dt_grid = {
        "clf__max_depth": [None, 5, 10], # 3 choices
        "clf__min_samples_split": [2, 5, 10], # 3 choices > 3x3=9
    }
    models["DT"] = (dt, dt_grid)
    # Random Forest
    rf = RandomForestClassifier(random_state=42, class_weight="balanced", n_jobs=-1)
    rf_grid = {
        "clf__n_estimators": [150, 300],
        "clf__max_depth": [None, 10],
        "clf__min_samples_split": [2, 5],   }
    models["RF"] = (rf, rf_grid)
    # SVM
    svm = SVC(probability=True, class_weight="balanced", random_state=42)
    svm_grid = {
        "clf__C": [0.5, 1.0, 2.0],
        "clf__kernel": ["rbf", "linear"],
        "clf__gamma": ["scale"],    }
    models["SVM"] = (svm, svm_grid)
    # XGBoost
    xgb = XGBClassifier(random_state=42, eval_metric="logloss", n_jobs=-1)
    xgb_grid = {
        "clf__n_estimators": [200, 400],
        "clf__max_depth": [3, 5],
        "clf__learning_rate": [0.05, 0.1],   }
    models["XGB"] = (xgb, xgb_grid)
    return models
```

```
def train_and_select_best(X, y):
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y    )
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    models = get_models_and_grids()
    best_name, best_estimator, best_score = None, None, -1
    for name, (clf, grid) in models.items():
        pipe = Pipeline([
            ("scaler", StandardScaler()),
            ("clf", clf)   ])
        gs = GridSearchCV(
            estimator=pipe,
            param_grid=grid,
            scoring="f1",
            cv=cv,
            n_jobs=-1,
            verbose=0    )
        gs.fit(X_train, y_train)
        y_pred = gs.predict(X_test)
        y_proba = gs.predict_proba(X_test)[:, 1]
        print("\n" + "="*70)
        print("Model:", name)
        print("Best params:", gs.best_params_)
        print("Best CV F1:", round(gs.best_score_, 4))
        print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
        print("Report:\n", classification_report(y_test, y_pred, digits=4))
        print("ROC-AUC:", round(roc_auc_score(y_test, y_proba), 4))
        if gs.best_score_ > best_score:
            best_score = gs.best_score_
            best_name = name
            best_estimator = gs.best_estimator_
```

```
best_model, best_name = train_and_select_best(X, y)


model_path = f"C:/Users/Noor/.spyder-py3/best_smartlock_model_{best_name}.joblib"
joblib.dump(best_model, model_path)
print("✅ Saved model to:", model_path)
import numpy as np
from collections import defaultdict


# Predict on ALL data
y_pred_all = best_model.predict(X)
y_proba_all = best_model.predict_proba(X)[:, 1]


results = defaultdict(list)


for i in range(len(y)):
    folder = meta.loc[i, "folder"]
    true_label = y[i]
    pred_label = y_pred_all[i]
    results[folder].append((true_label, pred_label))


print("="*70)
print("PER-FOLDER PERFORMANCE")
print("="*70)
```

```
for folder, vals in results.items():
    total = len(vals)
    correct = sum(1 for t, p in vals if t == p)
    wrong = total - correct

    false_accept = sum(1 for t, p in vals if t == 0 and p == 1)
    false_reject = sum(1 for t, p in vals if t == 1 and p == 0)

    acc = correct / total * 100

    print(f"\nFolder: {folder}")
    print(f"  Samples       : {total}")
    print(f"  Accuracy      : {acc:.2f}%")
    print(f"  Correct       : {correct}")
    print(f"  Wrong         : {wrong}")
    print(f"  False Accept   : {false_accept}")
    print(f"  False Reject   : {false_reject}")

# ============================
# ONE CELL: Upload -> preprocess -> feature -> predict -> decision
# ============================
import librosa
import numpy as np
import tkinter as tk
from tkinter import filedialog, messagebox
```

```
# ---- settings (must match training) ----
TARGET_SR = 16000
DURATION_SEC = 2.5
TOP_DB = 25         # silence trimming aggressiveness
THRESHOLD = 0.65     # 0.60~0.70


def _preprocess_external_like_training(wav_path, target_sr=TARGET_SR, duration_sec=DURATION_SEC, top_db=TOP_DB):
    # 1) load as mono + resample
    y, sr = librosa.load(wav_path, sr=target_sr, mono=True)

    # 2) trim leading/trailing silence
    y_trim, _ = librosa.effects.trim(y, top_db=top_db)
    if len(y_trim) >= 0.3 * len(y):
        y = y_trim

    # 3) normalize amplitude
    peak = np.max(np.abs(y)) + 1e-9
    y = y / peak

    # 4) pad/trim to fixed length
    target_len = int(target_sr * duration_sec)
    if len(y) > target_len:
        y = y[:target_len]
    else:
        y = np.pad(y, (0, target_len - len(y)))

    return y, target_sr
```

```
def upload_and_test():
    test_wav = filedialog.askopenfilename(
        title="Select WAV file",
        filetypes=[("WAV files", "*.wav")]
    )
    if not test_wav:
        return
    try:
        print("Test file:", test_wav)
        # ---- preprocess + extract features ----
        y_audio, sr = _preprocess_external_like_training(test_wav)
        feats = extract_features(y_audio, sr).reshape(1, -1)
        # ---- predict ----
        proba = float(best_model.predict_proba(feats)[0, 1])  # class 1 (AUTHORIZED)
        pred = int(proba >= THRESHOLD)
        # ---- decision ----
        print(f"Authorized probability: {proba:.4f} | threshold: {THRESHOLD:.2f}")
        if pred == 1:
            result = "✅ AUTHORIZED — Noor + correct phrase"
        else:
            result = "❌ NOT AUTHORIZED — wrong speaker or wrong phrase"
        messagebox.showinfo("Result", f"{result}\n\nProbability: {proba:.4f}\nThreshold: {THRESHOLD:.2f}")
    except Exception as e:
        messagebox.showerror("Error", str(e))
# ---- simple GUI window ----
root = tk.Tk()
root.title("Voice Verification")


btn = tk.Button(root, text="Upload WAV & Test", font=("Arial", 14), width=20, height=2, command=upload_and_test)
btn.pack(padx=25, pady=25)
root.mainloop()
```

4.4.1 Built Model Results:

```
In [1]: %runfile 'C:/Users/Noor/.spyder-py3/voice_recognition (1).py' --wdir
Noor_diff_99 → 99 wav files
Noor_open_20 → 21 wav files
Noor_open_85 → 85 wav files
Others_open_100 → 99 wav files
BASE_DIR: D:/anaconda_app/my_works/voices/voices
Folders found: ['Noor_diff_99', 'Noor_open_20', 'Noor_open_85', 'Others_open_100']
Others_open_100  -> 99 wav files
Noor_open_85     -> 85 wav files
Noor_open_20     -> 21 wav files
Noor_diff_99     -> 99 wav files
Total samples: 304
Class balance: {0: 198, 1: 106}
```

Figure 17: Check the files before starting.

```
======================================================================
Model: DT
Best params: {'clf__max_depth': None, 'clf__min_samples_split': 10}
Best CV F1: 0.8586
Confusion matrix:
 [[36  4]
 [ 5 16]]
Report:
              precision    recall  f1-score   support

           0     0.8780    0.9000    0.8889        40
           1     0.8000    0.7619    0.7805        21

    accuracy                         0.8525        61
   macro avg     0.8390    0.8310    0.8347        61
weighted avg     0.8512    0.8525    0.8516        61

ROC-AUC: 0.847
```

Figure 18-4: The results of Applying decision tree model.

```
======================================================================
Model: RF
Best params: {'clf__max_depth': None, 'clf__min_samples_split': 2, 'clf__n_estimators': 150}
Best CV F1: 0.9143
Confusion matrix:
 [[40  0]
 [ 4 17]]
Report:
              precision    recall  f1-score   support

           0     0.9091    1.0000    0.9524        40
           1     1.0000    0.8095    0.8947        21

    accuracy                         0.9344        61
   macro avg     0.9545    0.9048    0.9236        61
weighted avg     0.9404    0.9344    0.9325        61

ROC-AUC: 0.9869
```

Figure 19-4: The results of applying rain forest model.

```
======================================================================
Model: SVM
Best params: {'clf__C': 0.5, 'clf__gamma': 'scale', 'clf__kernel': 'linear'}
Best CV F1: 0.9882
Confusion matrix:
 [[39  1]
 [ 0 21]]
Report:
              precision    recall  f1-score   support

           0     1.0000    0.9750    0.9873        40
           1     0.9545    1.0000    0.9767        21

    accuracy                         0.9836        61
   macro avg     0.9773    0.9875    0.9820        61
weighted avg     0.9844    0.9836    0.9837        61

ROC-AUC: 1.0
```

Figure 20-4: The results of applying SVM on dataset.

```
Model: XGB
Best params: {'clf__learning_rate': 0.05, 'clf__max_depth': 3, 'clf__n_estimators': 200}
Best CV F1: 0.95
Confusion matrix:
 [[40  0]
 [ 4 17]]
Report:
              precision    recall  f1-score   support

           0     0.9091    1.0000    0.9524        40
           1     1.0000    0.8095    0.8947        21

    accuracy                         0.9344        61
   macro avg     0.9545    0.9048    0.9236        61
weighted avg     0.9404    0.9344    0.9325        61

ROC-AUC: 0.9976

✅ Best selected: SVM | CV F1: 0.9882
✅ Saved model to: C:/Users/Noor/.spyder-py3/best_smartlock_model_SVM.joblib
```

Figure 21-4: The result of applying XGBoost on the dataset (in the end it shows that SVM give us the highest accuracy).

```
Folder: Others_open_100
  Samples          : 99
  Accuracy         : 100.00%
  Correct          : 99
  Wrong            : 0
  False Accept     : 0
  False Reject     : 0

Folder: Noor_diff_99
  Samples          : 99
  Accuracy         : 98.99%
  Correct          : 98
  Wrong            : 1
  False Accept     : 1
  False Reject     : 0

Folder: Noor_open_85
  Samples          : 85
  Accuracy         : 100.00%
  Correct          : 85
  Wrong            : 0
  False Accept     : 0
  False Reject     : 0

Folder: Noor_open_20
  Samples          : 21
  Accuracy         : 100.00%
  Correct          : 21
  Wrong            : 0
  False Accept     : 0
  False Reject     : 0
```

Figure 22-4: The results of applying SVM on the 4 folders of dataset.

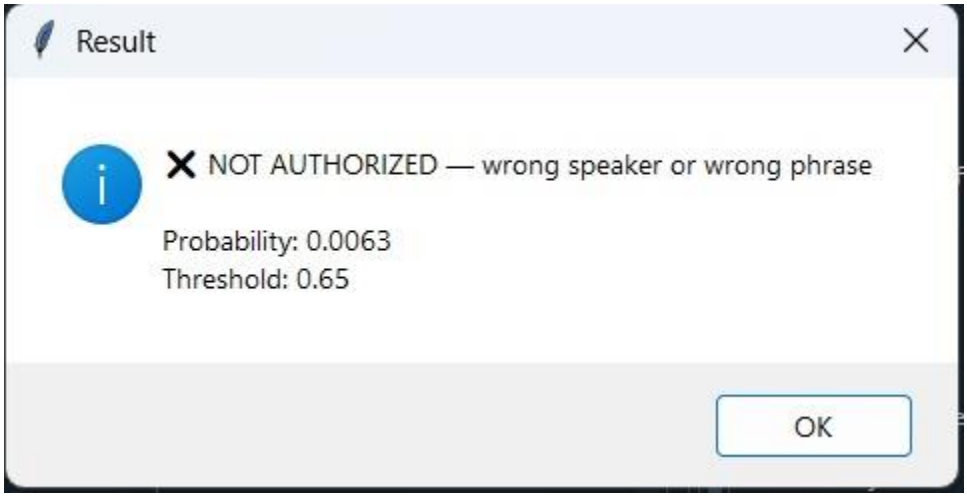


Figure 23-4: The sentence was (close the window) should be not authorized.

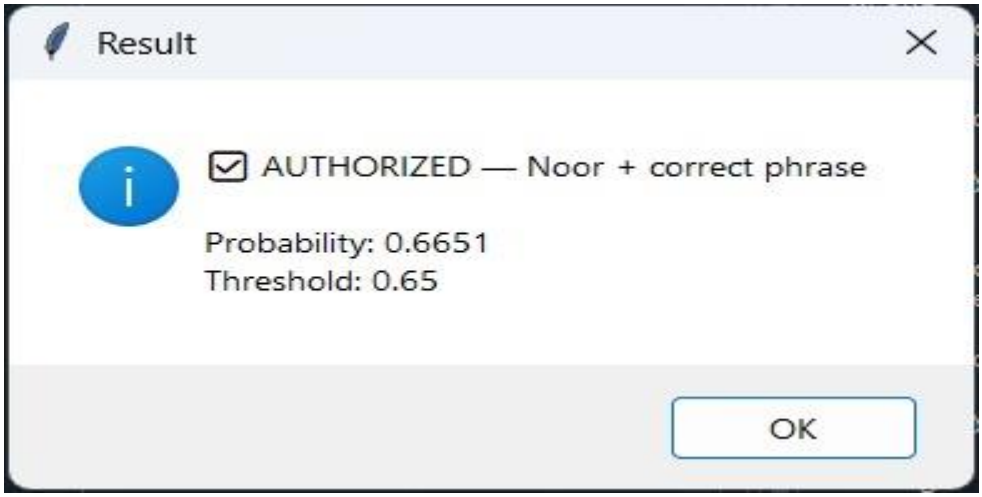


Figure 24-4: The sentence was (open the door) should be authorized.

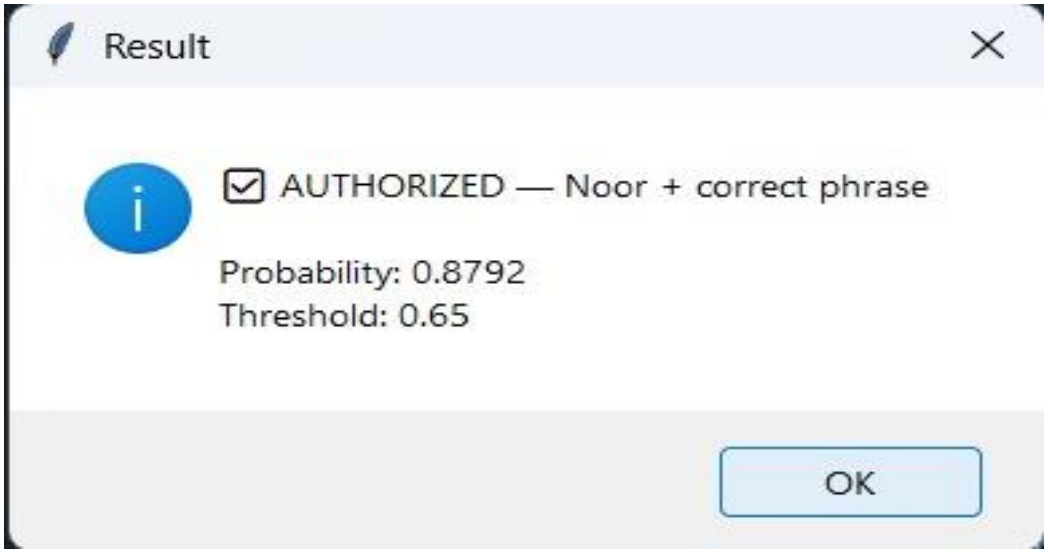


Figure 25-4: This one is also for the authorized.

Figure 26-4: This one for a non- authorized person with (open the door) sentence.

4.5.1 Pre - trained model results:

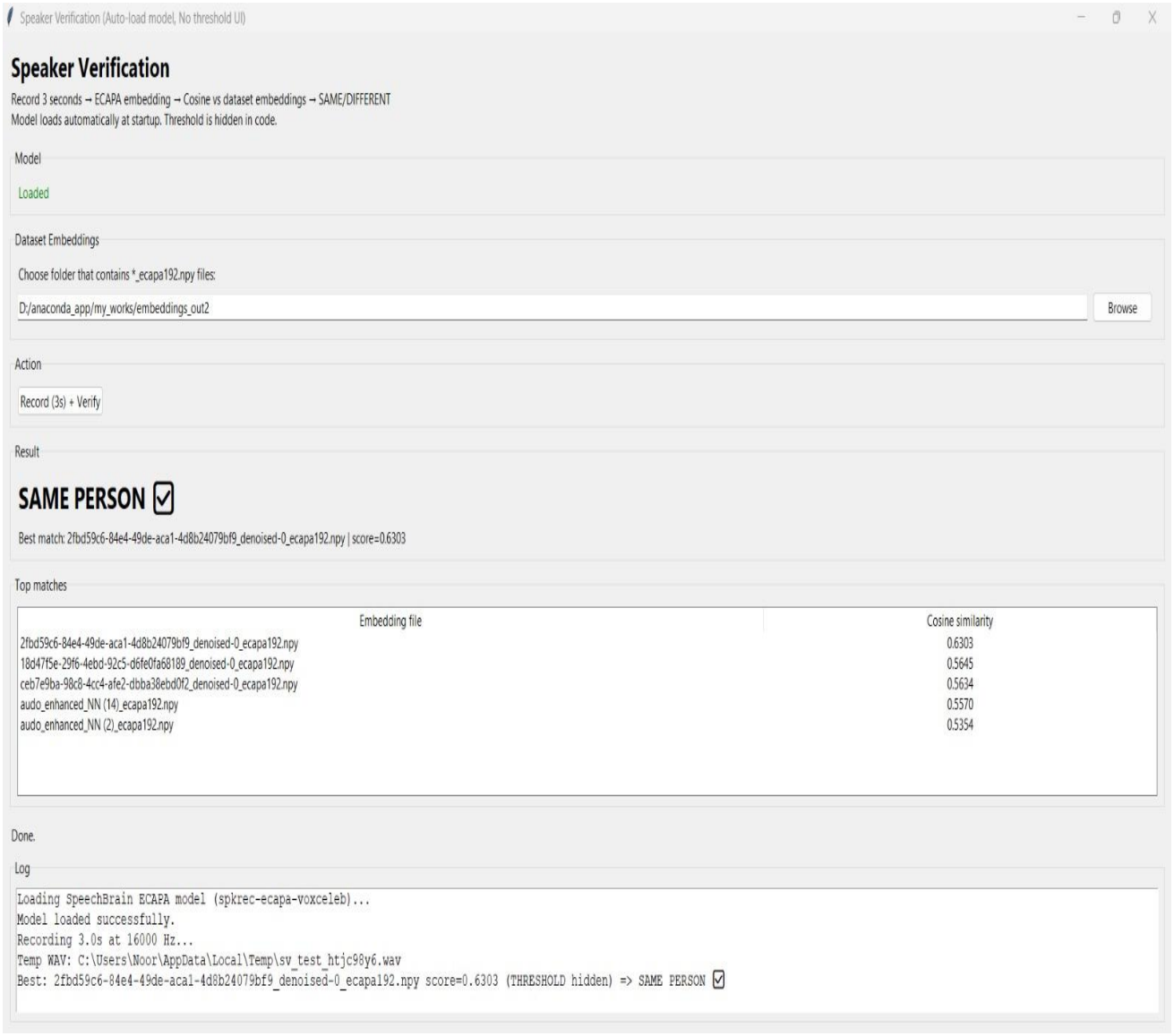


Figure 27-4: Pre-trained result of authorized person says (open the door).

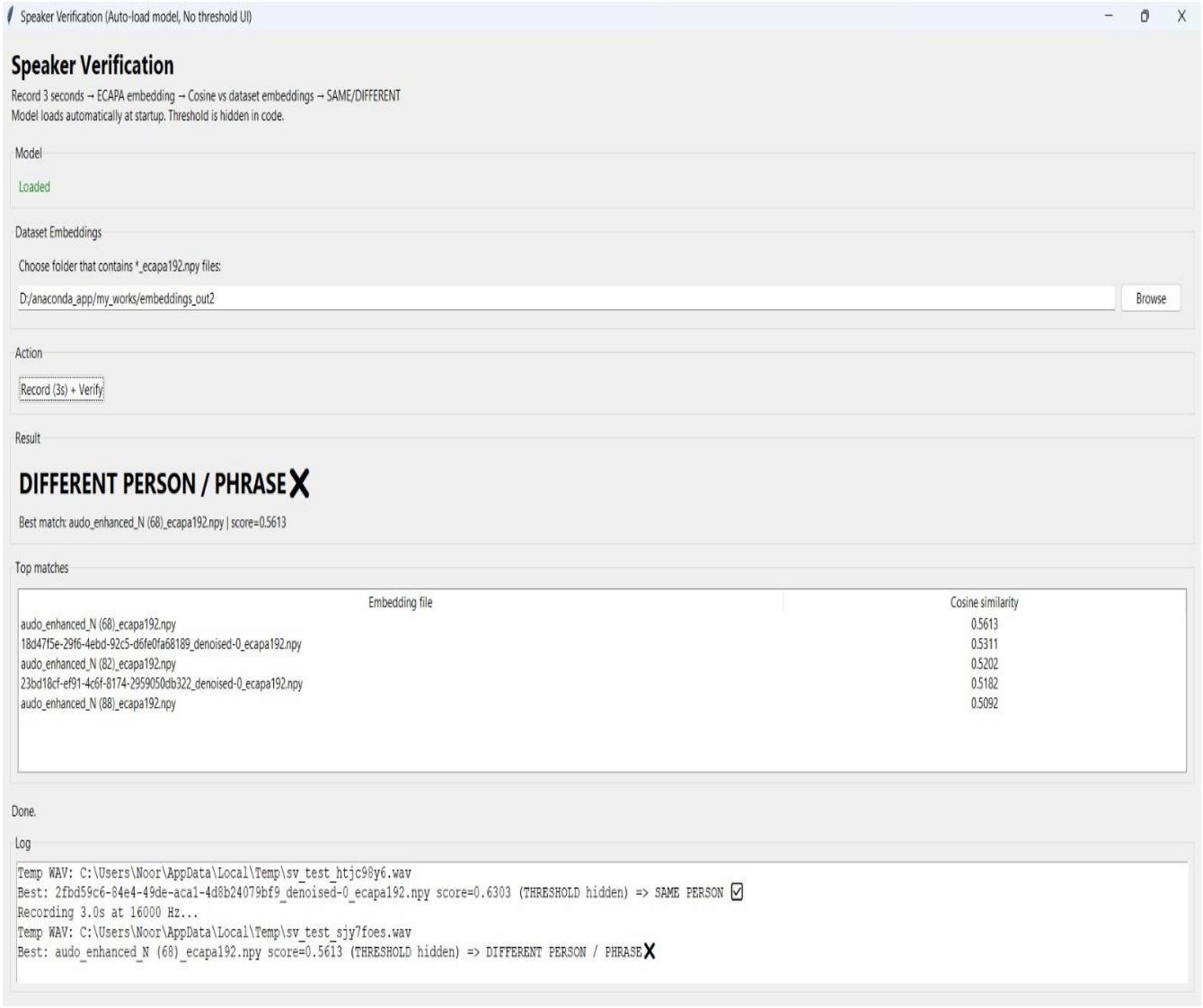


Figure 28-4: Pre-trained for the authorized says (close the window) sentence.

❖Chapter 5: Discussion

The results obtained from the implementation of the proposed voice-based door unlocking system demonstrate that voice recognition is a viable and effective biometric solution for access control applications. The analysis of Mel-Frequency Cepstral Coefficients (MFCCs) across multiple recording attempts showed strong intra-speaker consistency, confirming their ability to capture stable spectral characteristics that uniquely represent each speaker while remaining relatively insensitive to variations in speaking volume and minor environmental changes. In addition, pitch period and fundamental frequency measurements clearly reflected expected physiological differences among child, female, and male speakers, providing an additional discriminative layer that improved speaker differentiation. Formant frequency analysis further contributed to system accuracy by representing anatomical properties of the vocal tract, which remained consistent for each speaker despite natural speech variations. The applied preprocessing techniques, including band-pass filtering, Hamming windowing, and signal normalization, significantly enhanced signal quality by reducing background noise, minimizing spectral leakage, and standardizing amplitude levels, thereby improving feature reliability. Furthermore, the use of cosine similarity for feature comparison proved effective in measuring voice similarity independently of signal energy, while the integration of deep speaker embeddings enhanced robustness and generalization across recordings. Despite the promising performance under controlled conditions, the system may be affected by extreme noise levels, temporary voice changes, and limited scalability, highlighting the need for further optimization and extensive real-world testing to ensure long-term reliability and security.

❖Chapter 6: Conclusions and Recommendation

6.1 Conclusions

This project successfully designed and implemented a voice-based door access control system that utilizes speaker recognition as a biometric authentication method. By extracting and analyzing key speech features such as Mel-Frequency Cepstral Coefficients (MFCCs), pitch period, fundamental frequency, and formant frequencies, the system was able to accurately distinguish between authorized and unauthorized users. The experimental results demonstrated consistent performance across multiple recording attempts for different speakers, including child, female, and male voices, confirming the reliability of the selected features. The application of effective signal preprocessing techniques, including band-pass filtering, Hamming windowing, and normalization, significantly improved system robustness by reducing noise effects and ensuring stable feature extraction. Additionally, the use of cosine similarity and deep speaker embeddings enhanced matching accuracy and speaker discrimination. Overall, the findings confirm that voice recognition can serve as a secure, user-friendly, and practical solution for access control systems, particularly in smart and contactless environments.

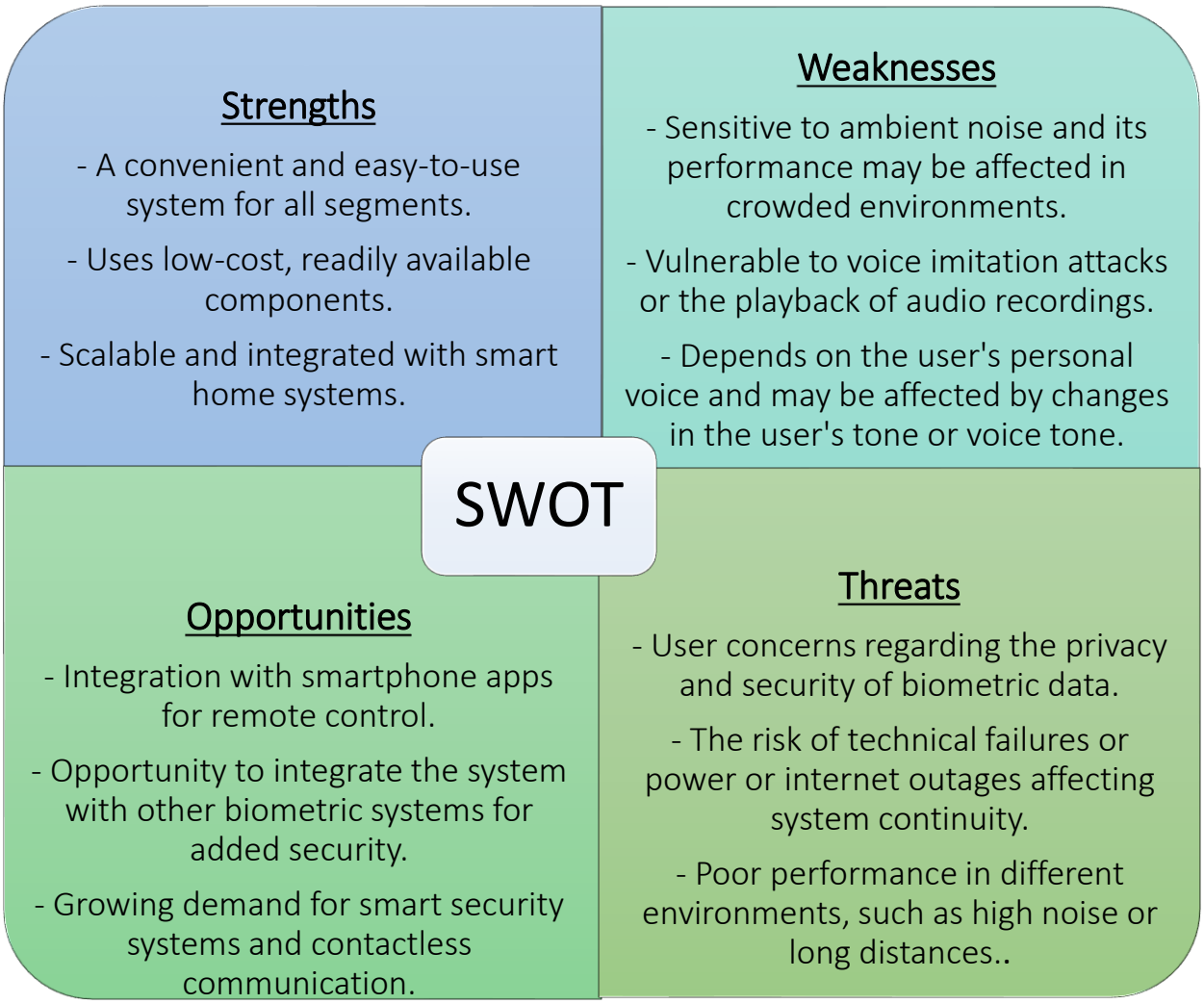
6.2 Recommendation

Based on the results of this project, it is recommended to enhance the proposed voice-based door access control system by increasing the number of enrolled users to evaluate scalability and improve system generalization. Implementing advanced noise reduction and voice activity detection techniques would further improve performance in noisy and real-world environments. Additionally, integrating multi-factor authentication, such as combining voice recognition with a PIN code or mobile verification, can significantly increase security. For practical deployment, optimizing the system for embedded hardware is recommended to reduce computational load and power consumption. Finally, continuous system updates and periodic re-enrollment of voice samples are advised to maintain accuracy and reliability over time as users' voices naturally change.

6.3 Future works

For future improvements, **Your Whisper Key–Sound ID Unlock** can be enhanced to perform reliably in noisy environments by integrating advanced noise reduction and more robust feature extraction methods. The system could be extended to support multiple users with secure voice profiles, including role-based access control. Mobile integration would allow users to manage profiles and receive notifications remotely, while continuous authentication could ensure security even after the door is unlocked. Hardware optimization can reduce size, cost, and power consumption, making the system more practical, and combining voice recognition with other biometric methods such as fingerprint or face recognition could provide multi-factor security. Additionally, implementing real-time feedback, access logging, and exploring advanced machine learning models, such as Transformer-based architectures, can further improve recognition accuracy and overall system performance.

❖Swot Analysis



❖References

1. https://ieeexplore.ieee.org/abstract/document/8743482
2. https://jontallen.ece.illinois.edu/uploads/537.F18/Book/main-all.pdf
3. https://citeseerx.ist.psu.edu/document?doi=b4e9c14c67b8aa431a40041cce0a3564144e1a2a&repid=rep1&type=pdf
4. https://www.ijirset.com/upload/2018/may/8_Speaker.pdf
5. https://new.eurasip.org/Proceedings/Ext/SPECOM2006/papers/021.pdf
6. https://web.ece.ucsb.edu/Faculty/Rabiner/ece259/Reprints/016_parallel%20processing%20pitch%20detector.pdf
7. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7965c68bedb3f7a7d566a1a1bb1d7a9e72c5a46a
8. https://assta.org/proceedings/sst/2006/sst2006-84.pdf
9. https://jjem.jnnce.ac.in/journals/SP-2/JJEMSP0231.pdf
10. https://www.akademiabaru.com/submit/index.php/ard/article/view/6137