ACOUSTIC SIGNAL IDENTIFICATION IN AN AUDIO TRACK

BY

SENG WEI XIANG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMPUTER ENGINEERING

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Information Technology (Honours) Computer Engineering at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to sincerely thank my supervisor, Mr. Lee Heng Yew, and my moderator, Dr Teoh Shen Khang, for providing me with the chance to learn more about speaker diarization and acoustic signal processing. Their invaluable guidance, insightful feedback, and continuous support were crucial throughout the development of this project. Whenever I encountered challenges or uncertainties, their advice and expertise were instrumental in helping me find solutions and move forward. A million thanks go to my supervisor and moderator for their patience and dedication.

In addition, I want to express my gratitude to my family and friends for their constant encouragement and support over my studies and during the completion of this final year project. Their understanding and motivation were a constant source of strength.

ABSTRACT

This project focuses on the identification and transcription of acoustic signals within audio tracks, specifically targeting multi-speaker English audio files without background noise or overlapping speech. The primary objective is to develop a standalone, local software program using Python that can reliably identify different speakers and produce transcriptions that are credited to each one without the need for an internet connection. The system employs techniques for audio signal processing, speaker diarization for segmenting the audio stream based on speaker identity, and automatic speech recognition for transcribing the spoken content, all implemented using local models and libraries. The methodology involves processing the input audio locally, applying speaker diarization to detect speaker changes and segment the audio, and subsequently transcribing each segment while associating it with the corresponding speaker, ensuring full offline operation. This project contributes to the field of audio analysis by creating a self-contained, offline-capable tool for speaker-aware acoustic signal processing and transcription in controlled environments, demonstrating the practical application of Python-based audio processing and machine learning tools that function independently of cloud services. The final output is a functional offline Python application capable of identifying speakers and generating speaker-labelled transcriptions for the specified audio constraints.

Area of Study: Audio Signal Processing

Keywords: Speaker Diarization, Speaker Transcription, Python Programming, Audio Processing, Signal Segmentation

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Hz	Hertz

LIST OF ABBREVIATIONS

ML	Machine Learning
DL	Deep Learning
SVM	Support Vector Machines
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
LTSM	Long Short-Term Memory
EEND	End-to-End Neural Diarization
GUI	Graphical User Interface
WER	Word Error Rate
DER	Diarization Error Rate
SAE	Speaker Attribution Error

Chapter 1
Introduction

This chapter introduces the background, motivation, contributions, and organization of the project. The development of automated acoustic signal identification systems is gaining importance due to the exponential growth in audio data across various fields. Acoustic signal identification is the process of analyzing and classifying sound signals to extract meaningful information, which is crucial in applications like speech recognition, environmental sound detection, and multimedia content analysis. Efficient identification of acoustic signals enables better management and utilization of audio data in many research and industrial domains [1].

1.1 Problem Statement and Motivation

The rapid increase in the volume of digital audio content presents significant challenges for manual audio analysis, which is time-intensive and prone to human error. Automated acoustic signal identification systems address this problem by providing fast, reliable, and scalable tools for extracting and classifying audio features [2]. Despite advancements in digital signal processing and machine learning, designing software that strikes a balance between accuracy, speed, and resource efficiency remains challenging.

One central difficulty lies in extracting robust and discriminative features from audio signals to uniquely identify acoustic events in diverse contexts. Feature extraction techniques are critical because they determine the quality of the representation used for matching and classification [3]. Moreover, efficient algorithms for comparing and matching these features are vital to ensure scalability, especially when dealing with large databases or real-time streams.

This project aims to develop a Python-based system that addresses these challenges through effective audio processing, feature extraction, and identification workflows. The goal is to reduce manual workload, improve recognition accuracy, and provide a flexible framework adaptable to future enhancements and real-world applications. By doing so, it contributes to the ongoing evolution of audio analysis technologies and supports broader efforts to harness the growing audio data landscape [1][2].

1.2 Objectives

The main objective of this project is to utilize existing powerful Python-based tools to achieve effective acoustic signal identification within audio tracks. The project applies state-of-the-art pre-trained models to process audio data, separating speakers and generating transcriptions.

The specific objectives include:

- Employing a pre-trained diarization tool to segment audio by speaker turns
- Using a robust speech-to-text model to transcribe segmented audio into text
- Integrating diarization and transcription results to accurately label audio content with speaker information
- Developing a software framework that combines these tools seamlessly to provide clear, structured outputs in a user-friendly manner

By leveraging well-established libraries, the project aims to build a reliable and scalable system that can automate speaker identification and transcription tasks efficiently, meeting the increasing demand for automated audio content analysis.

1.3 Project Scope and Direction

This project focuses on developing a local software system that processes English audio recordings without requiring an internet connection. The audio recordings used do not contain overlapping speech; speakers talk one at a time, simplifying the diarization process. The system is designed to run entirely offline, allowing users to maintain data privacy and operate in environments with limited or no internet access.

The scope includes:

- Applying speaker diarization to segment audio according to speaker turns, assuming no overlapping speakers
- Performing speech transcription on the segmented audio to produce accurate text results for English speech
- Combining these functionalities into a seamless local software solution
- Developing a user-friendly graphical user interface (GUI) to facilitate easy interaction with the software for users of varying technical abilities

The offline capability is achieved by using pre-downloaded pre-trained models and caching mechanisms, enabling the system to run independently from cloud services. This offline operation not only improves user data security but also reduces dependencies on external platforms, making the system robust and accessible for various applications involving English-language audio such as interviews, meetings, and transcription tasks where internet access may be restricted or undesirable.

1.4 Contributions

This project contributes by demonstrating the effective integration of powerful, pre-existing Python tools to automate acoustic signal identification and transcription tasks. By utilizing reliable pre-trained models, the project avoids the need to develop new algorithms from scratch, focusing instead on practical application and system integration.

Key contributions include:

- Delivering a complete local software solution capable of performing speaker diarization and transcription on English audio recordings without requiring internet connectivity, thus preserving user data privacy and enabling offline usage.
- Simplifying the process of audio analysis by integrating diarization and speech-to-text functions into a single, streamlined workflow, improving accessibility for users.
- Developing a user-friendly graphical interface that lowers the entry barrier for non-technical users, allowing easy interaction with the system's capabilities.
- Validating the approach on clean audio files without overlapping speech, demonstrating accurate segmentation and transcription results under these conditions.

These outcomes provide a foundation for further enhancements, including adaptation to more complex audio scenarios and real-time processing, and contribute practical value to the field of automated audio analysis by offering an accessible, scalable, and efficient toolset.

1.5 Report Organization

This report is organized into 7 chapters to provide a clear and systematic presentation of the project. Chapter 1 introduces the project background, problem statement, objectives, scope, contributions, and an overview of the report structure. Chapter 2 reviews relevant literature and existing solutions related to the project's domain, helping to establish the context and identify

gaps. Chapter 3 details the system methodology and approach, including design diagrams and descriptions that outline how the system is structured and operates. Chapter 4 covers the system design specifics, including block diagrams, component specifications, and how different parts of the system interact. Chapter 5 describes the system implementation process, hardware and software setup, configuration details, and includes screenshots and discussions of issues encountered during development. Chapter 6 focuses on system evaluation and discussion, presenting testing methods, performance results, project challenges, and an assessment of how well the objectives were met. Finally, Chapter 7 concludes the report with a summary of findings and provides recommendations for further work or improvements. This organization ensures a logical flow from conceptualization through design, development, testing, and evaluation, enabling readers to follow the comprehensive development lifecycle of the project.

Chapter 2

Literature Review

2.1 Existing Work in Acoustic Signal Processing

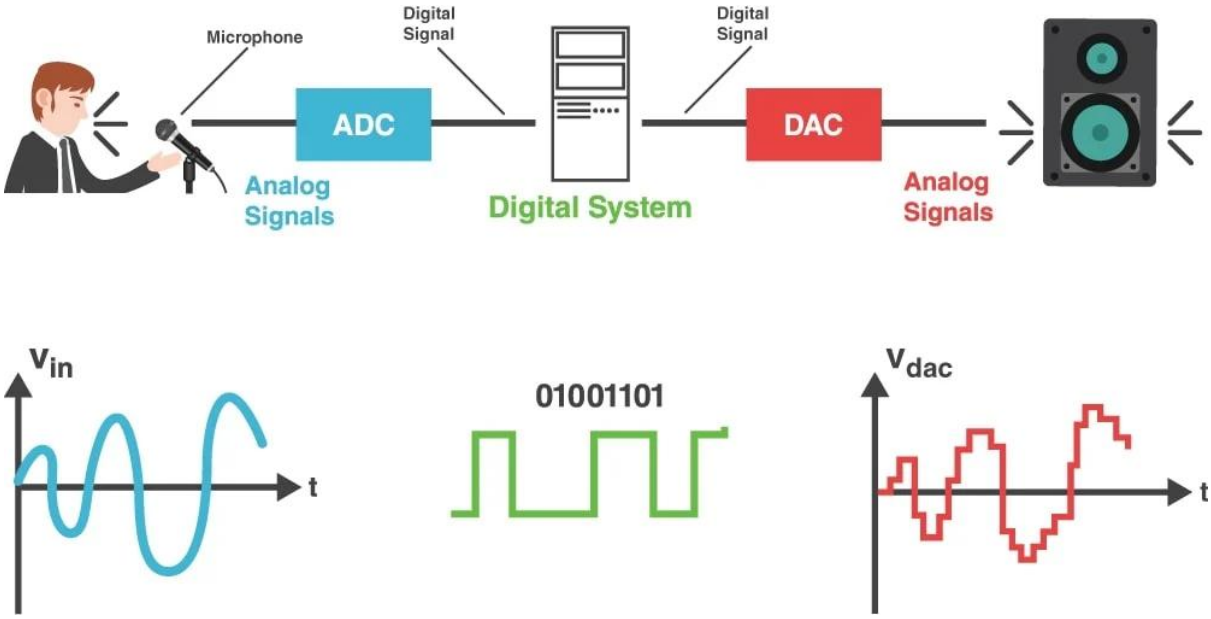


Figure 2.1 Signal Processing

Acoustic signal processing has undergone significant transformation since its inception, evolving from basic analog techniques to sophisticated modern methods incorporating machine learning and deep learning approaches.

2.1.1 Early and Traditional Signal Processing in Acoustics

Initially, acoustic signal processing relied heavily on analog methods and basic digital techniques for capturing and analyzing sound waves. Early systems used rudimentary analog sonobuoys and limited bandwidth telemetry to measure underwater or environmental acoustic signals. The fundamental operations consisted of noise reduction, signal enhancement, and segmentation to extract meaningful information from acoustic emissions (e.g., geological or marine environments) [4].

With the transition towards digital signal processing (DSP), methods such as Fourier transforms, cepstrum analysis, and wavelet transforms became prevalent. These methods provided tools for spectral analysis, time-frequency representation, and pattern detection that

enhanced acoustic data analysis fidelity and expanded applications to areas such as structural health monitoring, environmental acoustics, and mechanical fault diagnosis [5].

2.1.2 Advancements with Statistical and Model-based Approaches

Following DSP, model-based signal processing approaches were developed to exploit domain-specific knowledge about acoustic sources and propagation. Examples include acoustic color signature analysis and source separation techniques that aimed to isolate individual sound sources from mixed signals. These methods balance between statistical assumptions and physical modeling to optimize signal estimation and enhancement [6].

2.1.3 Integration of Machine Learning and Deep Learning

A major evolutionary step in acoustic signal processing is the incorporation of machine learning (ML) and deep learning (DL). These data-driven techniques automatically discern patterns and features from large acoustic data sets, enabling complex tasks that were difficult with classical methods—such as human speech recognition, acoustic scene analysis, and anomaly detection in rotating machinery and geological monitoring [7][8].

ML-based acoustic signal processing adopts a paradigm shift, moving from handcrafted feature engineering towards end-to-end learning models that can discover intricate relationships in data. For example, the use of hybrid approaches combines traditional DSP knowledge with deep learning for enhanced noise reduction, source separation, and dereverberation performance [7][8].

2.1.4 Recent Innovations and Multi-disciplinary Applications

Recent research extends signal processing techniques to various specialized applications including seismic damage monitoring using fractal analysis, infrasound detection of geological events, acoustic emission monitoring in material testing, and environmental impact assessments of underwater sound sources on marine life. The integration of multi-sensor arrays, real-time processing, and satellite data relays has expanded the scope and scale of acoustic monitoring [4][9].

Current research emphasizes the integration of deep learning with classical wave-based modeling for more accurate acoustic field reconstruction and holistic audio understanding.

There is ongoing exploration of multi-channel processing, spatial audio reproduction, and advanced feature extraction techniques to address the complexities of real-world acoustic environments [10][11].

2.2 Existing Work in Machine Learning for Acoustics

Machine learning (ML) has transformed acoustic signal processing across a wide spectrum of applications, advancing from traditional handcrafted feature techniques to sophisticated data-driven models that autonomously discover complex patterns in acoustic data. Initially, acoustic analysis relied on engineered features and classical signal processing methods. However, the advent of ML, particularly deep learning, has enabled unprecedented capabilities in the interpretation, classification, and modeling of acoustic phenomena such as human speech, environmental sounds, and mechanical vibrations [7][12].

2.2.1 Early and Traditional Approaches

Traditional acoustic signal processing focused on deterministic algorithms for noise reduction, feature extraction (e.g., spectral and cepstral coefficients), and classification. These approaches required extensive domain knowledge and manual tuning. Innovations in machine learning introduced algorithms such as Support Vector Machines (SVM), decision trees, and clustering methods, which improved pattern recognition in acoustic signals by learning from labeled datasets. This phase set the foundation for more adaptive and scalable acoustic models [13].

2.2.2 Emergence of Deep Learning

A notable shift in the field occurred with the introduction of deep learning architectures like Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Long Short-Term Memory (LSTM) networks. These models excel at automatically extracting hierarchical and temporal features from raw acoustic signals, outperforming classical models in tasks including speech recognition, sound event detection, and speaker identification. Deep learning methods facilitated handling variable-length contextual information and capturing reverberation and nonlinear acoustic effects [13][14].

2.2.3 Applications Across Domains

Machine learning in acoustics spans various sectors:

- Speech and Voice Processing: Enhanced speech recognition, speaker verification, voice biometrics, and language identification have benefited from ML models that can manage noisy and variable acoustic environments [15].
- Structural Health Monitoring: ML models analyze acoustic emission signals to classify damage types and monitor fatigue in materials like steel, enabling real-time diagnostics in engineering [16].
- Bioacoustics and Biodiversity: ML facilitates the automated classification of wildlife sounds, linking acoustic features to species identification and evolutionary analysis, providing insights into animal behavior and phylogeny [17].
- Underwater and Environmental Acoustics: Novel hybrid ML models combine acoustic and optical signals for improved localization and communication in challenging underwater environments [18].

2.2.4 Methodological Trends Applications Across Domains

Recent reviews highlight a trajectory towards deep learning-based methods combined with multisensor data fusion, automatic feature extraction, and advanced learning paradigms like transfer learning and few-shot learning. There is a growing emphasis on unsupervised and semi-supervised approaches to manage limited labeled data scenarios common in acoustic datasets. Ensemble learning strategies further enhance classification performance, as seen in detecting fake voice audio with high accuracy [13][19].

2.2.5 Emerging Challenges and Future Directions

Despite impressive progress, challenges remain in interpretability, data scarcity, robustness to noise and adversarial examples, and ethical considerations, such as deepfake audio detection. Novel frameworks that integrate physical constraints of acoustic wave propagation with ML models are emerging to improve interpretability and control. Research continues to address these challenges through hybrid models, improved feature engineering, and real-time adaptive algorithms [20][21].

2.3 Existing Work in Speaker Diarization

Figure 2.3 Speaker Diarization

Speaker diarization is the task that aims to answer the question "who spoke when?" in audio or video recordings by segmenting a conversation according to individual speaker identities [22][23]. The evolution of this field reflects advances in signal processing, machine learning, and deep learning techniques.

2.3.1 Early Approaches: Modular Systems and Clustering

Initial speaker diarization systems were primarily modular, featuring separate stages including segmentation, feature extraction, speaker embedding, and clustering [24]. The most common traditional approach was to cluster speaker embeddings, such as i-vectors or x-vectors, using algorithms like K-means or Gaussian Mixture Models. However, these methods often struggled with overlapping speech and determining the optimal number of speakers [22].

2.3.2 Incorporation of Evolutionary Computation Algorithms

Research explored optimization of speaker clustering using evolutionary computation techniques like Genetic Algorithms, Particle Swarm Optimization, Differential Evolution, and Teaching-Learning-Based Optimization. These methods aimed to improve clustering accuracy by optimizing cluster numbers and grouping criteria, particularly in broadcast news and heterogeneous audio sources [22].

2.3.3 Introduction of Deep Learning and Neural Approaches

The advent of deep learning brought revolutionary changes by enabling end-to-end neural diarization models. These models replaced multi-stage pipelines with unified architectures that

learn to assign speaker labels directly from audio inputs, improving performance especially in complex scenarios with overlapping speakers [23][25][26].

2.3.4 End-to-End Neural Diarization (EEND)

EEND formulated diarization as a multi-label classification problem, allowing simultaneous detection of multiple active speakers. This method uses permutation-free objective functions to minimize diarization errors and understands speaker overlaps better than clustering methods [25][26]. Recent advances have integrated EEND with vector clustering to handle real conversational speech effectively, addressing challenges like arbitrary number of speakers and overlapped speech [27].

2.3.5 Joint Modeling and Sequence-to-Sequence Neural Diarization

Novel architectures apply sequence-to-sequence frameworks to perform both online and offline diarization, incorporating automatic speaker detection and better speaker representation. These approaches are pushing the boundaries by integrating diarization with other speech tasks such as speech recognition for comprehensive audio understanding [28][29].

2.3.6 Context-Aware and Prediction-Based Models

Recent studies have investigated the use of contextual information and conversation state prediction to enhance diarization. Techniques using Long Short-Term Memory (LSTM) networks, Markov Chains, and distance metrics on speaker-specific contextual similarity show promise in predicting speaker states and diarization labels in natural conversations without manual tagging [30][31].

2.3.7 Emerging Trends: Multimodal and Audio-Visual Speaker Diarization

As multimedia content grows, combining audio with visual cues like faces for speaker diarization is gaining attention. Audio-visual diarization improves robustness and accuracy by leveraging synchronized visual information alongside audio, facilitating applications in video analysis and multimedia content management [32].

2.4 Existing Work in Speaker Transcription

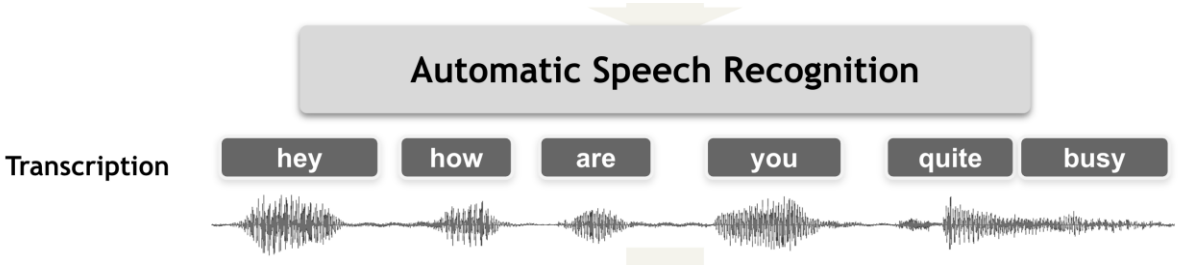


Figure 2.4 Speaker Transcription

Speaker transcription is the process of converting spoken language from audio recordings into written text. This transformation enables the structured representation of unstructured audio data, making it accessible for search, analysis, and documentation across a wide range of applications such as interviews, meetings, podcasts, and legal proceedings [33].

2.4.1 Early Development

Speech recognition originated in the 1950s with primitive systems that recognized simple vocabulary, such as the ten digits of English. Early systems used rule-based methods and template matching algorithms, focusing mostly on isolated word recognition. By the 1980s, advances in faster recognition algorithms and mathematical modeling enabled speaker-independent and continuous speech recognition over larger vocabularies. This period also saw the advent of Hidden Markov Models (HMM), which became a foundational technique due to their ability to model speech variability statistically [34].

2.4.2 Statistical Modeling and Hidden Markov Models

HMMs dominated the speech recognition landscape in the late 20th century, providing the ability to handle temporal variability and noise. Coupled with Gaussian Mixture Models (GMM) for acoustic modeling and n-gram language models, systems became more robust and applicable to real-world applications. This era also witnessed the scaling of systems to support continuous speech and larger vocabularies with reasonable accuracy [35].

2.4.3 Machine Learning and Neural Networks

Machine learning techniques started to supplement and partially replace traditional HMM-GMM frameworks. The rise of Deep Neural Networks (DNNs), especially with the introduction of algorithms like Deep Belief Networks and Convolutional Neural Networks,

brought significant accuracy improvements. These models better captured complex acoustic patterns and contextual speech dependencies than HMMs alone. This period also enabled the integration of speech recognition into consumer devices and services such as smartphones and GPS systems [36].

2.4.4 Deep Learning and End-to-End Systems

Speech recognition underwent a revolutionary leap due to deep learning advances. End-to-end neural architectures such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and more recently Transformer models and wav2vec have dramatically improved performance. These systems reduce reliance on handcrafted features and complicated pipelines, simultaneously learning acoustic, pronunciation, and language models. Usage has become widespread in virtual assistants (Amazon Alexa, Apple Siri), voice search, transcription, and translation applications, enabling natural, hands-free interaction [36][37][38].

2.4.5 Large-Scale and Weakly-Supervised Models

Large-scale models like OpenAI Whisper use weakly supervised learning on extensive audio datasets covering multiple languages and domains. These models demonstrate impressive transcription capability for single and multi-speaker audio, enhancing robustness to noise, accents, and domain variations. However, handling long audio segments requires advanced buffering or sliding window methods, and timestamp accuracy remains a challenge for long-form transcription [39].

2.4.6 Emerging Trends and Future Directions

Recent research highlights the integration of ASR with broader AI interfaces for more natural, hands-free communication. Innovations in silent or subvocal speech recognition address privacy and accessibility limitations. Cultural and linguistic diversity in speech recognition remains an area needing more focus to enhance inclusivity. The transition to machine listening—extracting meaning beyond just words—points toward evolving applications in digital humanities and cross-linguistic communication [40].

Chapter 3

System Methodology/Approach

This chapter outlines the methodology and approach applied in the development of the standalone software for **acoustic signal identification in an audio track**. The methodology includes the design process, architecture, and workflow that enable the system to perform diarization and transcription of audio files.

3.1 System Design Diagram

This project adopts a modular design approach to enable audio diarization and transcription in a standalone desktop software. The methodology is divided into four major components:

1. **Audio Input Module** – accepts pre-recorded English audio files. The expected input must have minimal background noise and **no overlapping speakers** to ensure accurate diarization and transcription.
2. **Speaker Diarization Module** – utilizes *pyannote.audio* to segment the audio and assign speaker labels (e.g., Speaker 1, Speaker 2, Unknown) [41] [42] [46].
3. **Speech Recognition Module** – employs *OpenAI Whisper* to generate transcriptions of each segment [43] [44].
4. **Graphical User Interface (GUI)** – developed in Python using Tkinter, which allows users to upload audio, start processing, and view diarization results with transcription after processing is completed.

The system is designed to support **sequential single-file analysis**. Users can process one audio file at a time. Once the results are shown, they may either upload another file for a new analysis or exit the program.

3.1.1 System Architecture Diagram

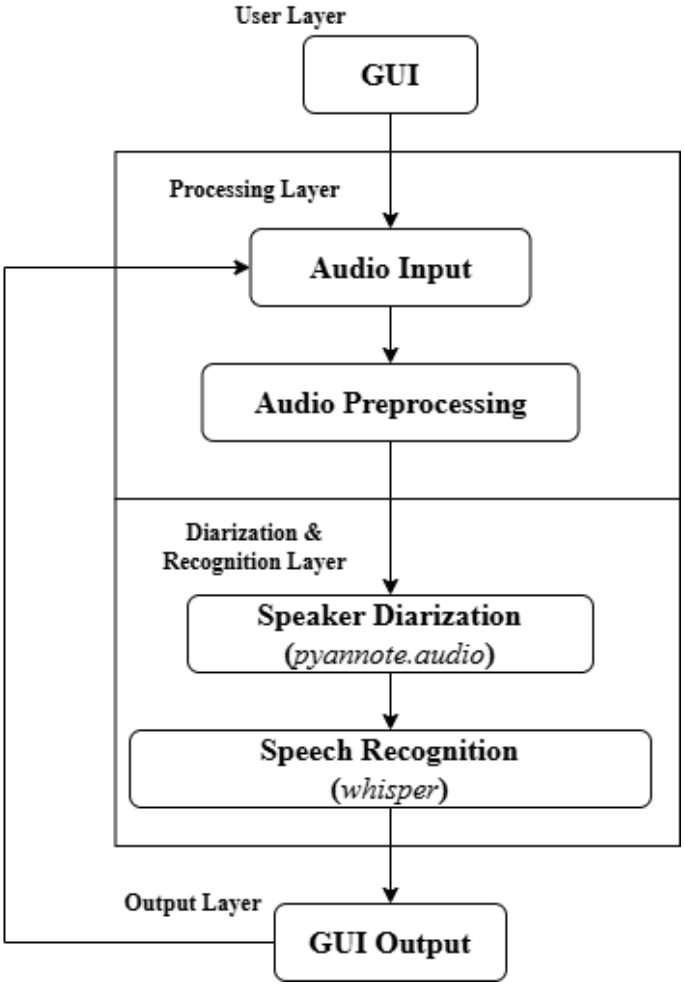


Figure 3.1.1 System Architecture Diagram

The architecture of the system consists of four interconnected layers:

- **User Layer**: Provides audio input and interacts with the software through the GUI.
- **Processing Layer**: Handles signal preprocessing, including resampling and temporary file handling.
- **Diarization & Recognition Layer**: Uses *pyannote.audio* for speaker segmentation and *Whisper* for transcription [41-44].
- **Output Layer**: Displays results in the GUI after processing. Users can then decide whether to continue with another audio file or exit.

3.1.2 Use Case Diagram and Description

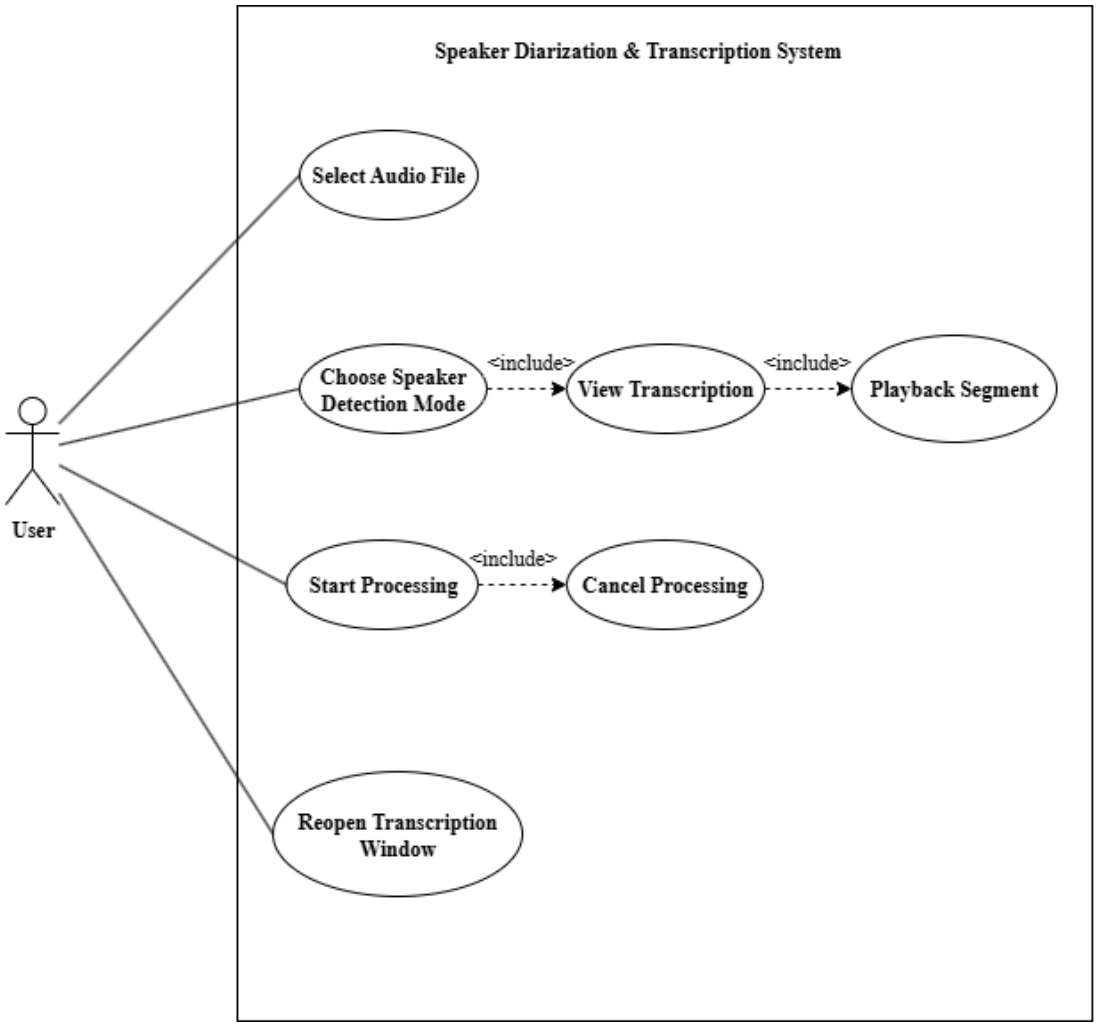


Figure 3.1.2 Use Case Diagram

Actors

1. **User** – interacts with the software to select audio files, configure speaker detection, start processing, and view results.

Use Cases

1. **Select Audio File** – User chooses an audio file (MP3, WAV, FLAC, OGG).
2. **Choose Speaker Detection Mode** – User selects **auto** or **manual** speaker number **before starting**.
3. **Start Processing** – System performs diarization and transcription.
4. **Cancel Processing** – User can cancel the ongoing process.
5. **View Transcription** – User views the processed transcription with speaker labels.

- **Playback Segment** (sub-function) – User can play a specific segment by double-clicking in the transcription window.

6. **Reopen Transcription Window** – User reopens the last transcription window.

Explanation

- The **User** is the sole actor.
- The **primary flow**: select a file → start processing → view transcription.
- Optional flows include **cancel processing**, **choose auto/manual speaker mode**, **playback segments**, and **reopen transcription**.

3.1.3 Activity Diagram

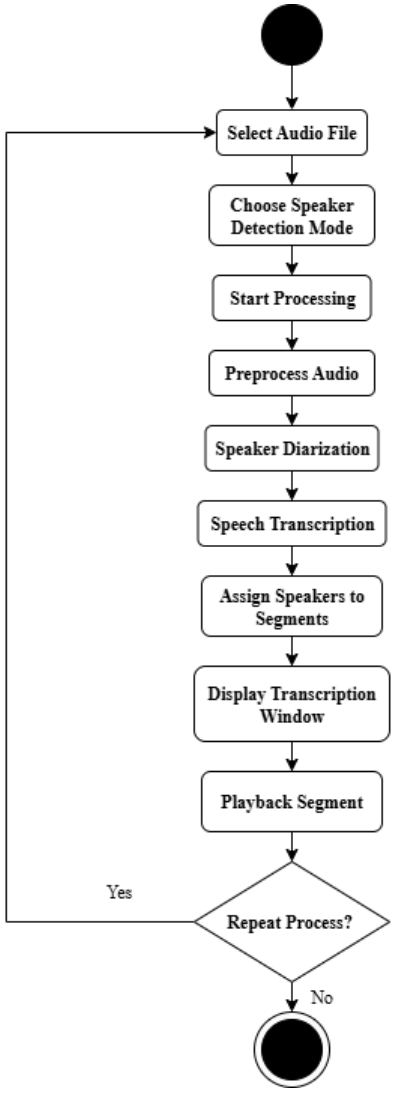


Figure 3.1.3 Activity Diagram

The workflow is as follows:

1. Start

2. Select Audio File – User chooses an audio file (MP3, WAV, FLAC, OGG).

3. Choose Speaker Detection Mode – User selects **auto** or **manual** speaker number.

4. Start Processing – System begins analysis.

5. Preprocess Audio

- Convert to mono if necessary
- Resample audio to 16 kHz
- Save to temporary WAV file

6. Speaker Diarization (PyAnnote) [41] [42] [46]

- Detect speakers automatically or use manual input
- Generate speaker-labeled segments

7. Speech Transcription (Whisper) [43] [44]

- Transcribe English speech
- Generate timestamped transcription segments

8. Assign Speakers to Transcription Segments

- Match transcription segments to diarization segments
- Label unmatched segments as "Unknown"
- Extend last segment to match full audio duration

9. Display Transcription Window

- Show speaker-labeled transcription in scrollable text box
- Enable **playback of** segments within the window (double-click)

10. Repeat Process? – User decides whether to analyze another audio file

- Yes → Go to Step 2 (Select Audio File)
- No → Proceed to Step 11

11. End

Decision Points:

- If **no audio file selected** → prompt user to select a file.
- If **cancel requested** during processing → terminate current process and go to Step 10 (Repeat Process decision).

Chapter 4

System Design

This chapter describes the detailed design of the standalone software for acoustic signal identification in an audio track. It provides all necessary information for someone to understand the program structure, components, and their interactions. The chapter covers system block diagrams, component specifications, and interaction operations.

4.1 System Block Diagram

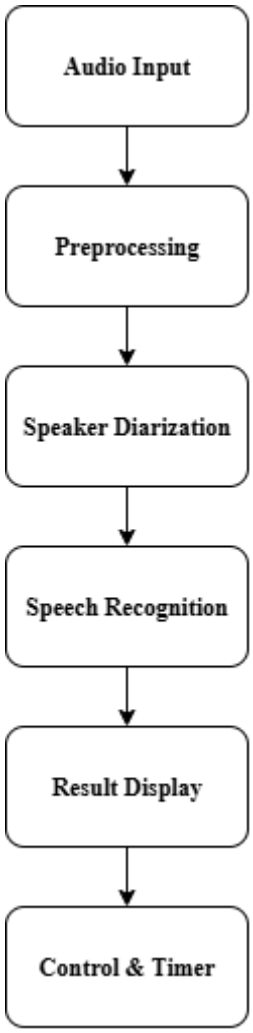


Figure 4.1 System Block Diagram

The system consists of the following major components:

1. **Audio Input Module** – handles the selection of audio files. Acceptable formats include .mp3, .wav, .flac, and .ogg. Only one file is processed at a time.

2. **Preprocessing Module** – performs signal processing tasks such as converting audio to mono, resampling to 16 kHz, and saving to a temporary file for further analysis.
3. **Speaker Diarization Module** – uses *pyannote.audio* to segment the audio and assign speaker labels.
4. **Speech Recognition Module** – uses *OpenAI Whisper* to transcribe each segment.
5. **Result Display Module (GUI)** – developed in Python with Tkinter. Displays diarization results with transcriptions after processing is complete. Users may then choose to process another audio file or exit the program.
6. **Control & Timer Module** – manages the GUI interactions, process timing, and cancellation requests.

4.2 System Components Specifications

The system modules are described as follows:

1. Audio Input Module

- Accepts audio files: .mp3, .wav, .flac, .ogg.
- Only one file can be processed at a time.
- Libraries used: tkinter, filedialog, os.

2. Preprocessing Module

- Converts stereo audio to mono.
- Resamples audio to 16 kHz.
- Saves a temporary WAV file for further processing.
- Libraries used: torchaudio, torch, tempfile [45-47].

3. Speaker Diarization Module

- Uses **pyannote.audio v3.1** to segment audio into speaker turns.
- Assigns speaker labels: Speaker 1, Speaker 2, or Unknown.
- Works with non-overlapping English audio with minimal background noise.
- Libraries used: pyannote.audio, torch [41] [42] [46].

4. Speech Recognition Module

- Uses **OpenAI Whisper (turbo)** for transcription.
- Processes each diarized segment individually.
- Preserves start and end times for each segment.
- Libraries used: whisper, torch [43] [44].

5. Result Display Module (GUI)

- Built with Tkinter.
- Displays speaker labels and transcriptions in a scrollable text widget.
- Supports segment playback with current line highlighted.
- Allows the user to **reopen the transcription window** to view previous results without reprocessing the audio file.
- Libraries used: tkinter, pygame [50] [51].

6. Control & Timer Module

- Manages timers and user cancellation requests.
- Ensures sequential workflow and prevents multiple concurrent processes.
- Libraries used: time, threading [48] [49].

4.3 Software Architecture and Module Design

The modules interact in a clearly defined sequence:

- **Modular Design:** Each module operates independently but communicates through defined inputs/outputs. This ensures maintainability and allows updates or replacements without affecting other modules.
- **Data Flow:** Audio File → Preprocessing → Diarization → Transcription → GUI Output
- **Control Flow:**
  - GUI triggers preprocessing
  - Preprocessing feeds diarization
  - Diarization results guide transcription
  - GUI displays final output
  - Only one file is processed at a time; after completion, the user may start a new session
- **Error Handling:**
  - Validates file type
  - Handles empty audio or failed processing
  - Ensures safe stopping if the user cancels mid-process
- **Dependencies/Libraries:**
  - All external Python libraries are listed in 4.2
  - Maintains software environment for Windows 11 with Python 3.12.5

- **Advantages:**
    - Modularity enables easy debugging, scaling, and future improvements (e.g., adding overlapping speaker support or new transcription models)

4.4 System Components Interaction Operations

1. **Audio Selection:** User selects a single audio file. GUI validates the file type and enables the start button.
2. **Preprocessing:** Audio is converted to mono, resampled to 16 kHz, and saved as a temporary file.
3. **Speaker Diarization:** Pyannote processes the audio and returns a list of segments with speaker labels [41] [42] [46].
4. **Transcription:** Whisper transcribes each segment, producing text with start and end times [43] [44].
5. **Result Display:** GUI shows diarization and transcription in a scrollable window. Users can play audio segments; the current line is highlighted.
6. **Reopen Transcription Window:** Users can reopen the transcription window to view previous results without starting a new process.
7. **Repeat Operation:** After results are displayed, the user may select a new audio file and repeat the process. Only one file is processed at a time.
8. **Timer & Control:** A timer shows elapsed processing time. Users can cancel processing before completion, which safely stops the workflow.

Chapter 5

System Implementation

This chapter describes in detail how the standalone software for acoustic signal identification is developed and implemented. The chapter provides information on the hardware and software setup, configuration, system operation, challenges encountered, and concluding remarks. All necessary details are provided for someone to reproduce the system.

5.1 Hardware Setup

The hardware used for this project is a **single personal computer**, which handles all processing tasks. No additional hardware devices are required, as the project is entirely software-based.

Table 5.1 Specifications of computer

Description	Specifications
Processor	Intel Core i5-13500
Operating System	Windows 11
Graphic	NVIDIA GeForce RTX 4060
Memory	32GB DDR5 RAM
Storage	1TB SATA HDD

All audio processing, diarization, transcription, and GUI operations are executed on this machine.

5.2 Software Setup

The software is implemented using **Python 3.12.5**, and the code is written using **Visual Studio Code** as the development environment. Required libraries and tools include:

- **Tkinter** – for the graphical user interface
- **Pyannote.audio v3.1** – for speaker diarization [41] [42] [46]
- **OpenAI Whisper (turbo)** – for transcription of audio segments
    - **Note:** Whisper requires **FFmpeg** to process audio files. In this project, FFmpeg was installed using the **Chocolatey package manager** (*choco install ffmpeg*). After installation, FFmpeg must be added to the system PATH [43] [44].
- **Torchaudio and Torch** – for audio preprocessing and handling [45-47]

- **Pygame** – Pygame is used for playback and also prevent software crashes while playing audio segments in the GUI [50] [51].
- **Tempfile** – for handling temporary audio files
- **Threading / Time** – Threading is used to prevent the GUI from freezing while running long tasks, such as diarization and transcription. Timers track process duration [48] [49].

All Python libraries are installed using pip. Once all dependencies are installed, the software can be executed offline.

5.3 Setting and Configuration

Before running the software:

1. Ensure Python 3.12.5 is installed and added to the system PATH.
2. Install required Python libraries:
   pip install torch torchaudio pyannote.audio openai-whisper pygame
3. Install **FFmpeg** using Chocolatey [44]:
   choco install ffmpeg
4. Ensure FFmpeg is accessible in the system PATH by opening a terminal and typing:
   ffmpeg -version
5. Place the Python script and any required configuration files in a single project folder.

5.4 System Operation

This section describes the operation of the software for acoustic signal identification, including speaker diarization, transcription, and playback of audio segments.

5.4.1 Main GUI

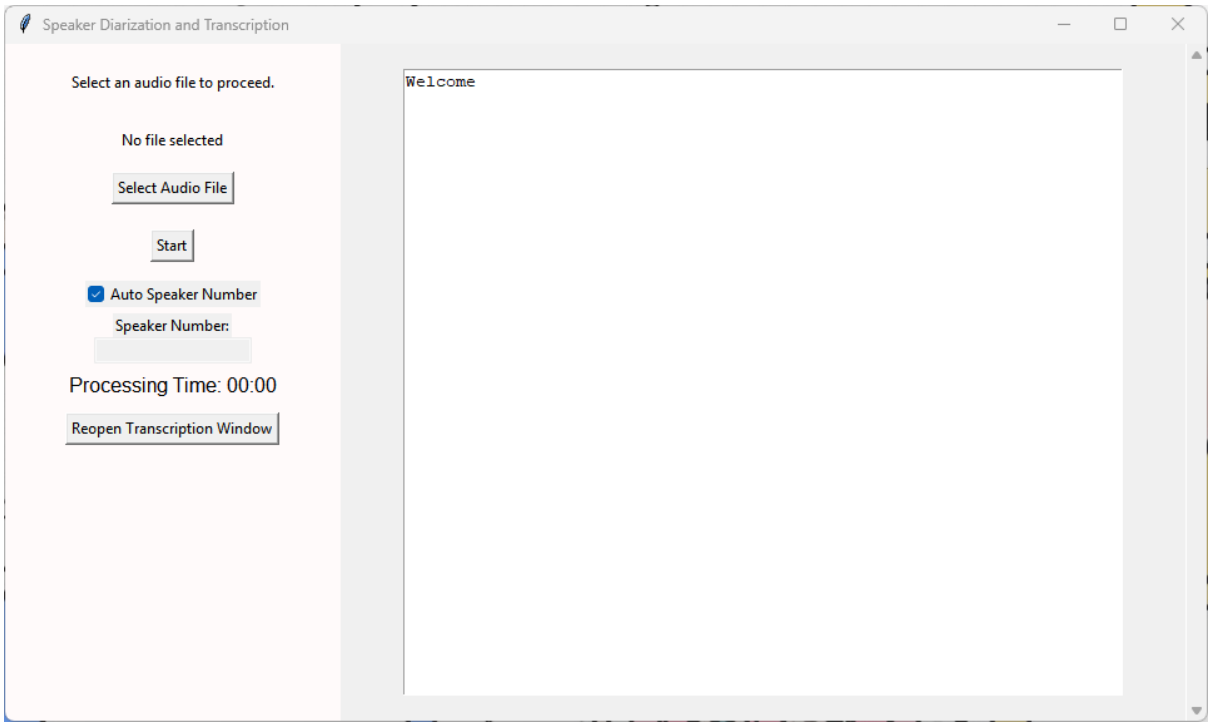


Figure 5.4.1 Main GUI

- The software is launched by running the Python script, it is built using **Tkinter**, providing a user-friendly graphical interface for selecting audio files, starting process, and viewing results.
- The main window provides the following options and features:
    - **Select Audio File** – choose an audio file to process.
    - **Start/Cancel button** – initially labeled **Start**.
        - Clicking **Start** begins the processing pipeline.

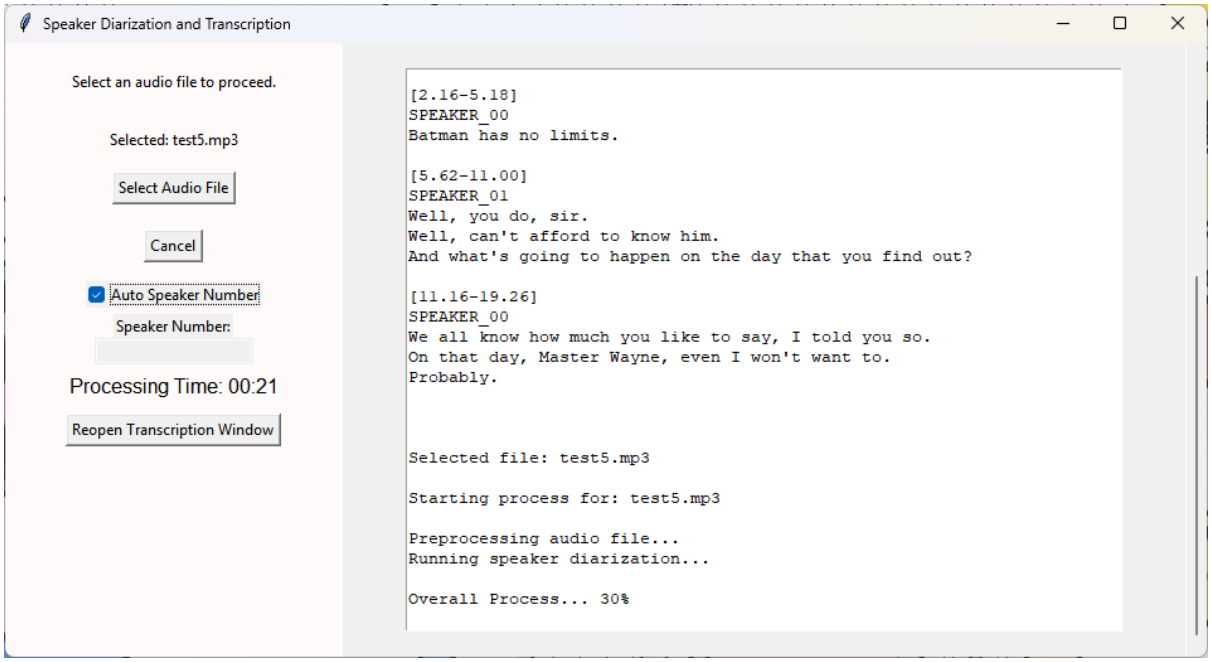


Figure 5.4.2 Main GUI Cancel button while running the task

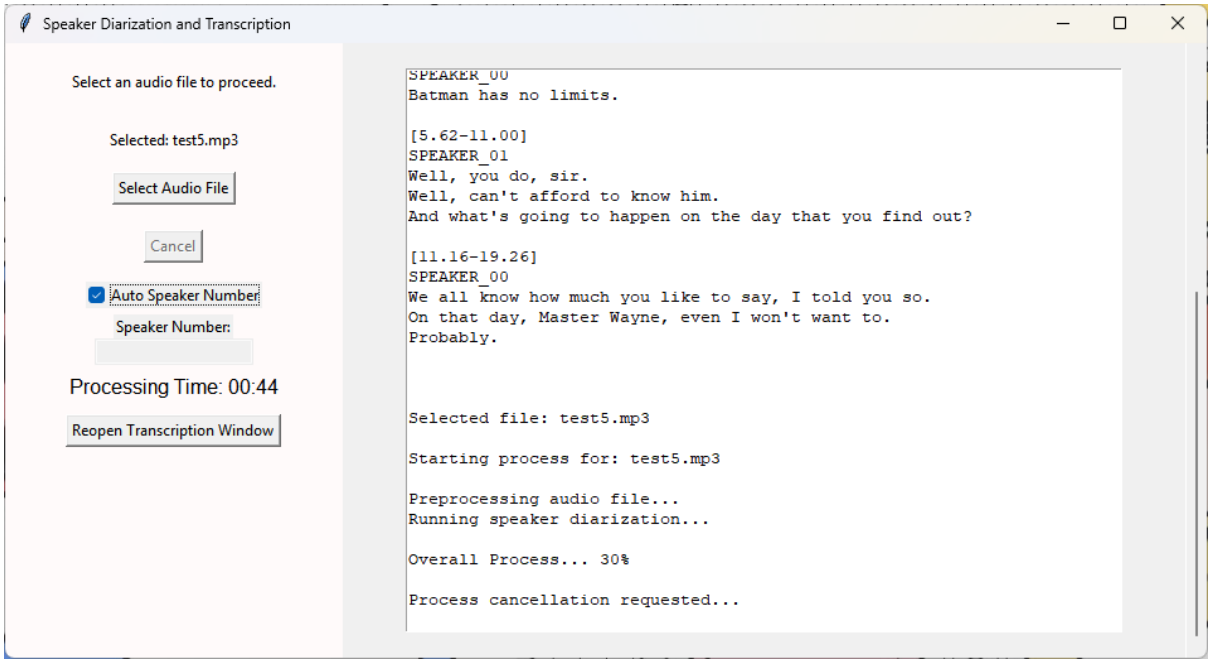


Figure 5.4.3 Cancel Pending after the button pressed

- While processing, the button changes to **Cancel**, allowing the user to safely stop the current task (**Cancel Pending**).

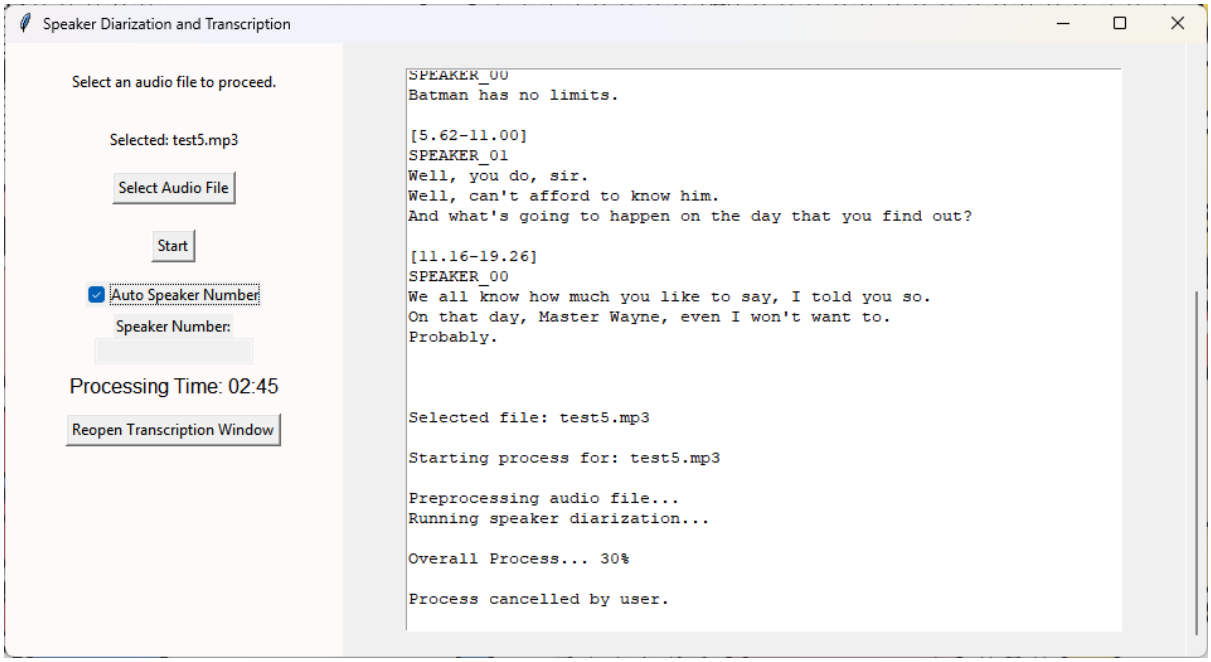


Figure 5.4.4 Cancel Completed

        - After the process stops safely, the button resets to **Start** (**Cancel Completed**).
    - **Speaker Detection Mode** – choose **Auto Speaker Number** or **Manual Speaker Number** (manual input required).
    - **Processing Timer** – shows the duration of the current processing task.
    - **Reopen Transcription Window button** – opens a separate window to view the most recent transcription without reprocessing the audio file.
    - **Scrollable text area** – displays real-time logs, progress updates, and results.
- The GUI ensures **only one file is processed at a time**, it is also responsive and prevents freezing during long tasks by using **threading**.
- After processing is complete, the user can choose to **repeat the process** with another audio file or **exit the program**.

5.4.2 Audio File Selection

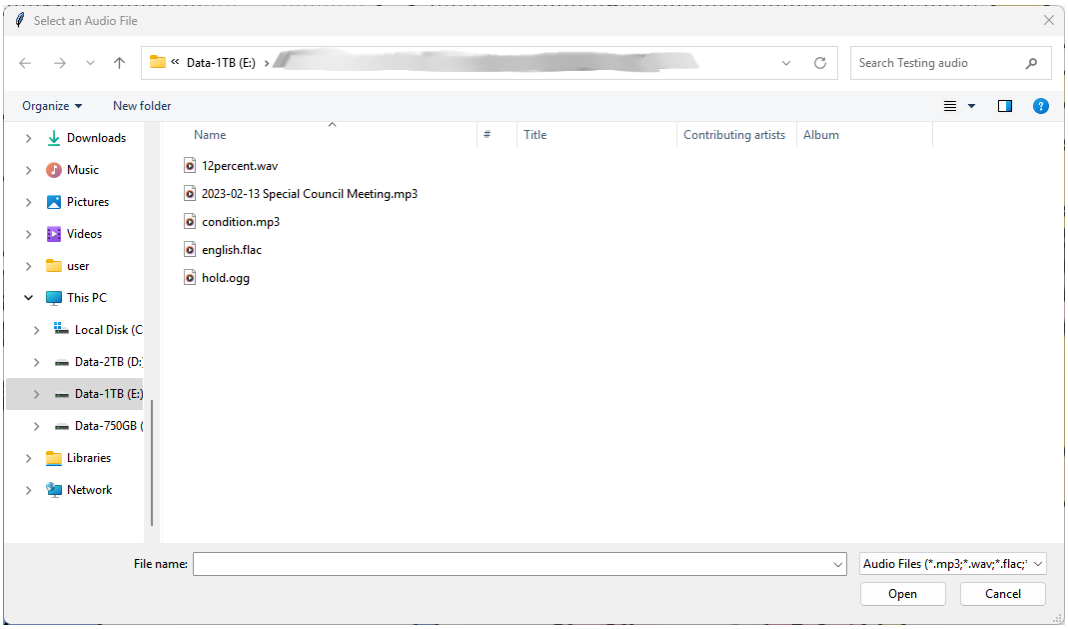


Figure 5.4.5 Select Audio File

- The user selects an audio file using the **"Select Audio File"** button.
- Supported formats include *.mp3*, *.wav*, *.flac*, and *.ogg*.
- The selected file name appears in the GUI, confirming the selection.

5.4.3 Speaker Detection Mode

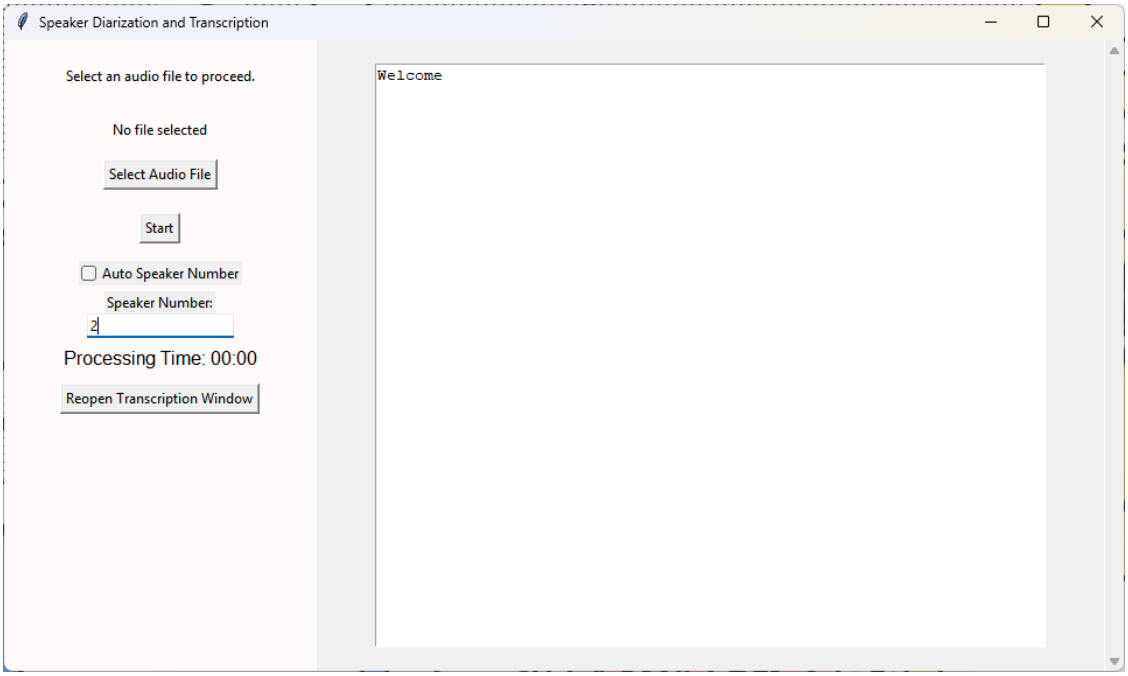


Figure 5.4.6 Speaker Detection Mode (Checkbox Not Selected)

- **Auto Speaker Number** checkbox:
    - **Default state** : ticked (enabled). The system automatically detects the number of speakers. Refer to **Figure 5.4.1** in Section 5.4.1 for the GUI layout.
    - **Manual input field behavior**: when the checkbox is ticked, the entry field for specifying speaker number is **disabled**.
    - If the user **unticks** the checkbox, the entry field becomes **editable**, allowing manual entry of the number of speakers.
    - **User input validation**: if a non-numeric value is entered, the system will **prompt an error**. Detailed behavior and error message are described in Section **5.4.6 Error Handling**.
- This choice must be made **before starting the processing**, ensuring consistent results.

5.4.4 Processing Workflow

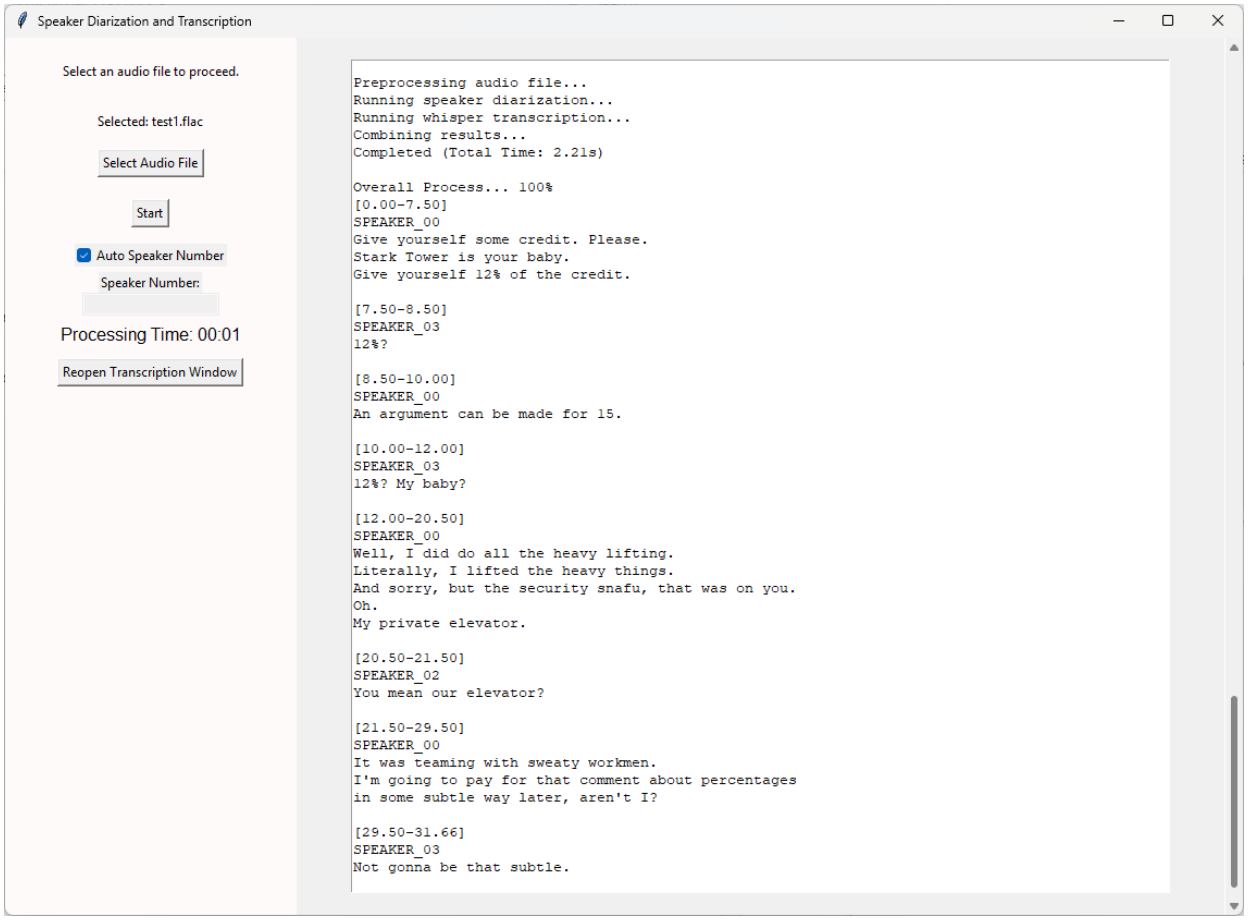


Figure 5.4.7 Result in main GUI

Once the **Start** button is clicked with a valid audio file selected:

i. Preprocessing Module [47] [48]

- When the user clicks **Start**, the selected audio file undergoes preprocessing:
  - **Convert to mono** if the audio is stereo.
  - **Resample to 16 kHz** for compatibility with Pyannote and Whisper [41-43].
  - **Save to a temporary WAV file** for further processing.
- Preprocessing ensures consistent audio input for diarization and transcription.

ii. Speaker Diarization Module [41] [42] [46]

- The preprocessed audio is sent to the pre-trained Pyannote model *speaker-diarization-3.1* for speaker diarization.
- Depending on the speaker detection mode, the model either detects the number of speakers automatically or uses the manually provided number.
- The model first detects speech regions using **voice activity detection (VAD)**.
- Speech regions are converted into speaker embeddings, which capture the characteristics of each speaker's voice.
- Embeddings are clustered to identify distinct speakers, and the start/end times of each cluster are used as timestamps.
- The module outputs a list of segments with **speaker labels** and **start/end times**.

iii. Speech Transcription Module [43] [44]

- Each diarized segment is transcribed using **OpenAI Whisper (turbo)**.
- Whisper processes the audio waveform with a pre-trained neural network and produces the corresponding text.
- The output includes timestamped transcription segments in English.

iv. Assigning Speakers to Transcription Segments

- Transcription segments are matched to the diarization segments using **overlap-based assignment**.
- Segments that cannot be matched are labeled as **"Unknown"**.
- The last transcription segment is **extended to match the full audio duration**, ensuring complete coverage.
- Progress updates and elapsed time are displayed in the GUI.

5.4.5 Result Display and Playback

- The results are displayed in a **scrollable text window** with speaker labels and transcriptions. (Main GUI Refer to Figure 5.4.7)

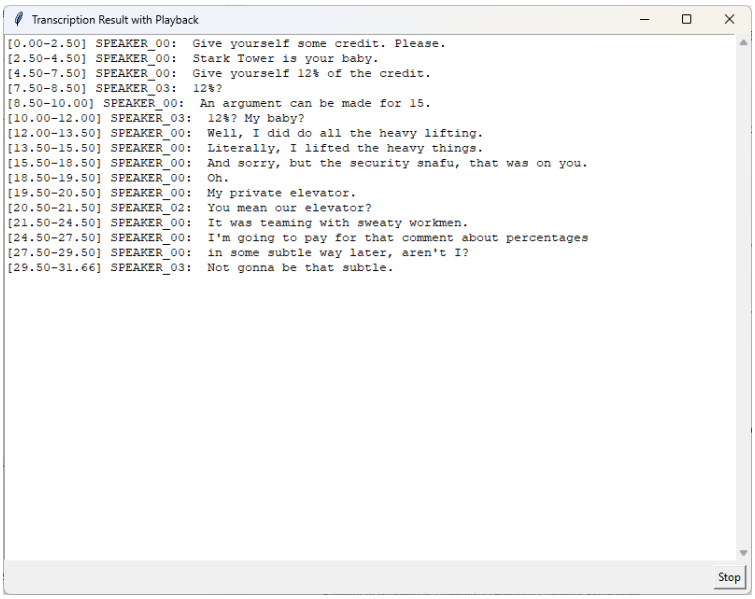


Figure 5.4.8 Transcription Window

- Upon completion of processing, the **transcription window automatically pops up**, showing the speaker-labeled transcription.

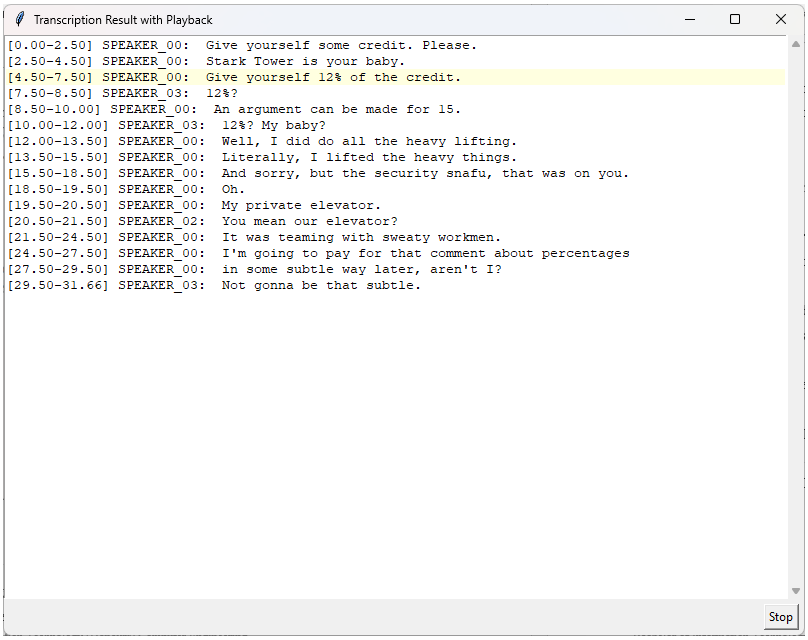


Figure 5.4.9 Transcription Window Playback

- Users can **double-click a segment** to play it, and the **current line is highlighted** during playback.
- The **Reopen Transcription Window** allows users to view the latest transcription without reprocessing the audio file.
- Playback is handled using **Pygame** for compatibility [50] [51].
- After viewing the results, the user can:
    - **Select another audio file** and repeat the process.
    - **Exit the software** by closing the main window.

5.4.6 Error Handling

The software provides safeguards for common user actions:

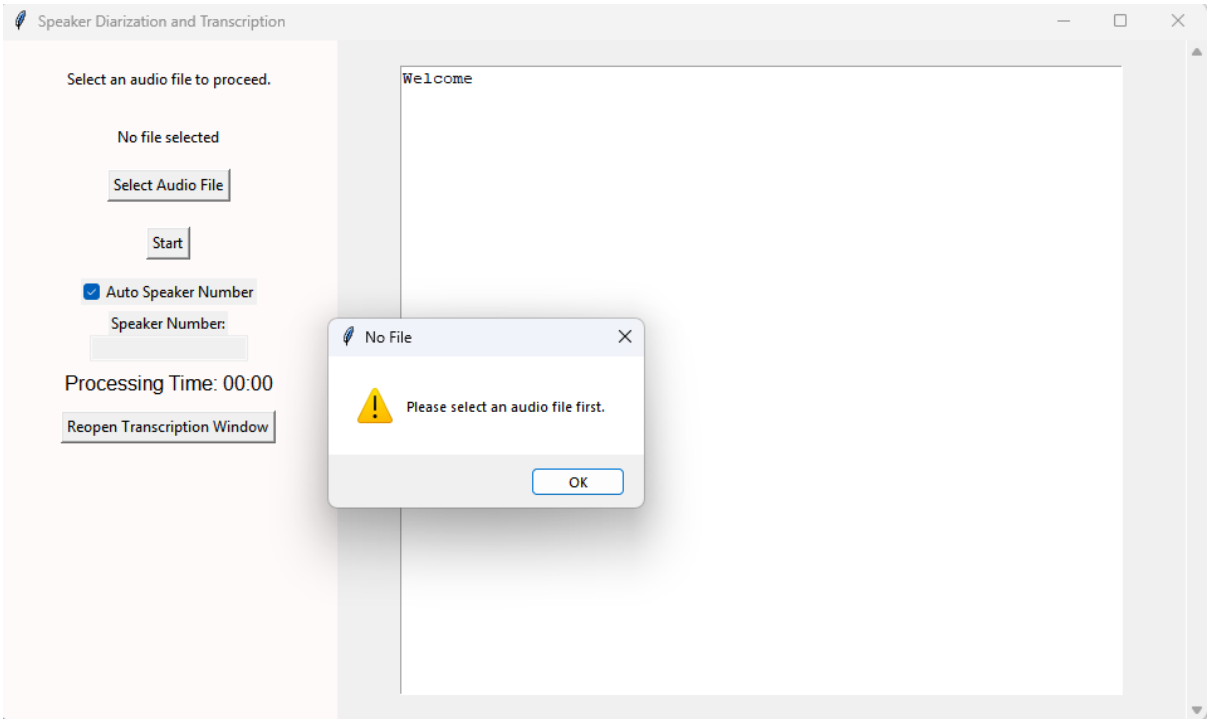


Figure 5.4.10 No Audio File Selected

- **No Audio File Selected:**
    - If the user clicks **Start** without selecting an audio file, a warning message is displayed.
    - The process cannot begin until a valid file is chosen.

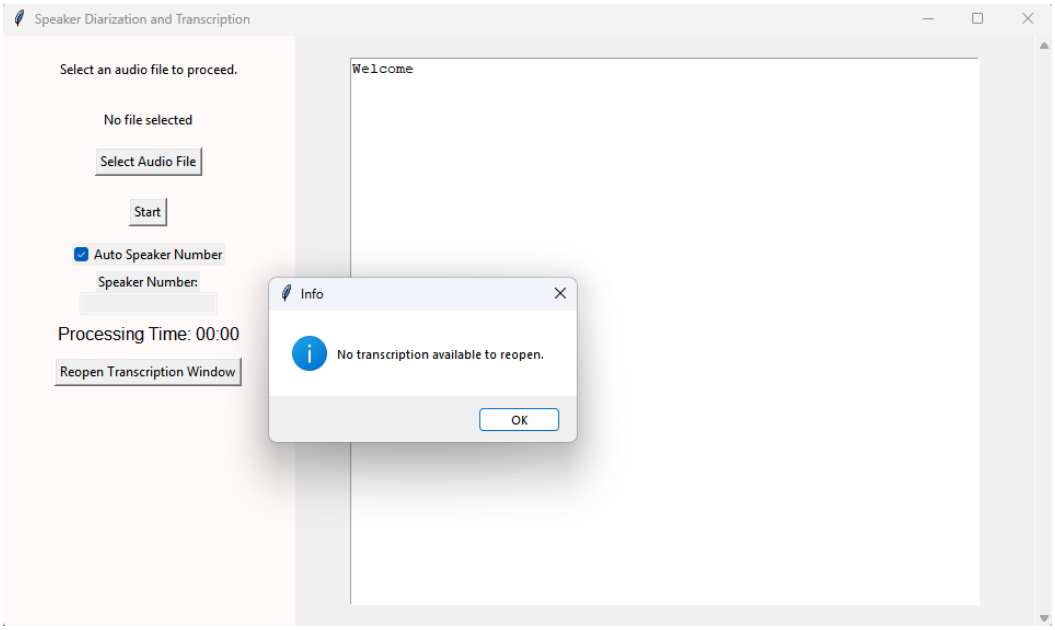


Figure 5.4.11 Opening Transcription Window Before Task Execution

- **Reopen Transcription Window Without Prior Results:**
  - If the user clicks **Reopen Transcription Window** before any transcription has been completed, an information message is displayed indicating that no transcription results are available.

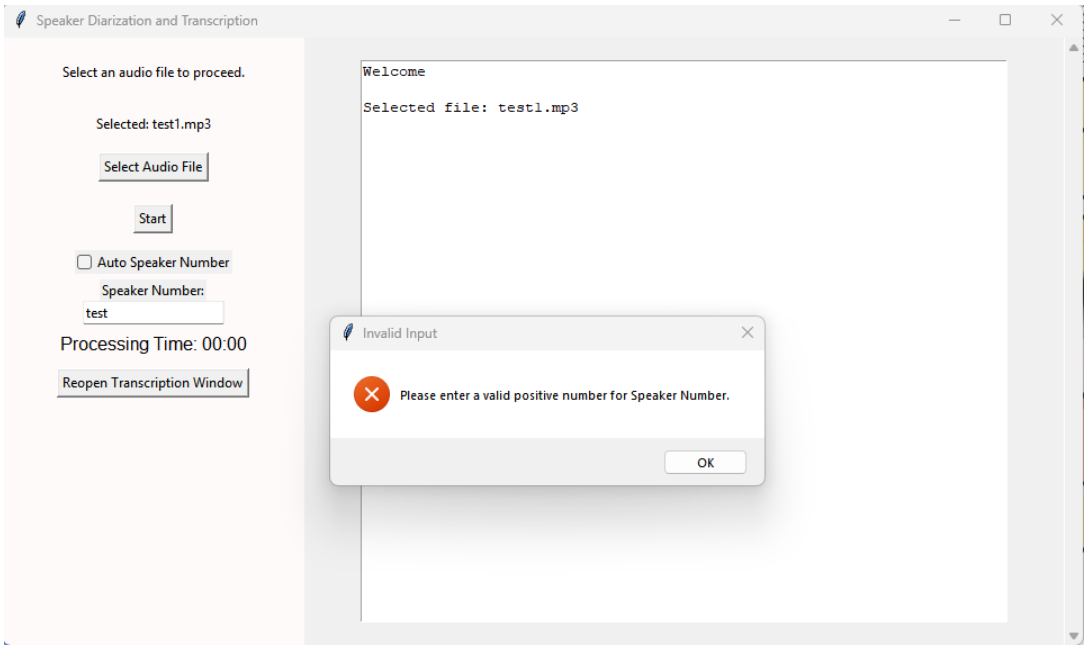


Figure 5.4.12 Invalid Speaker Number

- **Invalid Manual Speaker Input:**
    - If the user enters a non-numeric value in the manual speaker number field, the system displays an **error dialog** and prevents processing until a valid numeric value is provided.

These measures prevent users from starting processes without valid input or attempting to view results that do not exist.

5.5 Implementation Issues and Challenges

During the development of the **Acoustic Signal Identification** software, several challenges were encountered. One major challenge was ensuring accurate speaker diarization with PyAnnote.audio, especially for short or unclear segments. Segments that could not be confidently labeled were assigned as **"Unknown"** to maintain consistency.

Another challenge was integrating diarization results with Whisper transcription. Careful handling of segment start and end times was required to match transcription segments to speaker labels accurately. Preprocessing audio, such as converting stereo to mono and resampling to 16 kHz, also required attention to avoid potential audio quality loss or file access errors.

A significant implementation issue was maintaining **GUI responsiveness**. Without threading, the GUI would freeze while running long tasks like diarization or transcription. This was resolved by using **threading**, allowing the GUI to remain responsive and enabling features such as timers, cancellation, and playback highlighting.

Another consideration was **FFmpeg** integration, which is required by Whisper to process audio files. Installing FFmpeg via Chocolatey and ensuring it was accessible in the system PATH was necessary for proper transcription functionality.

Additionally, user input validation and proper handling of process cancellation were implemented to avoid errors and allow the user to safely repeat or exit the process. Overall, despite these challenges, the software successfully integrates all modules to provide a functional standalone application for acoustic signal identification.

5.6 Concluding Remark

The development of the **Acoustic Signal Identification** software successfully demonstrates the integration of multiple Python libraries and modules to create a standalone application. Through careful design and implementation, the software can perform audio preprocessing, speaker diarization, and transcription while providing a responsive and user-friendly GUI.

Key achievements include:

- Accurate diarization and transcription of English audio with minimal background noise.
- A GUI that supports segment playback, highlighting, and the ability to reopen transcription results without reprocessing.
- Robust handling of user interactions, including cancellation requests and input validation.
- Stable audio playback with Pygame.
- Seamless integration of FFmpeg for audio processing with Whisper.

Despite challenges such as maintaining GUI responsiveness and coordinating multiple modules, the final software demonstrates reliable performance for single-file audio analysis. The modular design also allows for future enhancements, such as support for overlapping speech, multi-language transcription, or additional audio formats.

Overall, Chapter 5 shows that the software is fully functional and provides a solid foundation for both practical use and further development in acoustic signal identification applications.

Chapter 6

System Evaluation and Discussion

This chapter evaluates the performance of the standalone software for acoustic signal identification in pre-recorded audio files. It discusses system testing, performance metrics, challenges encountered, and an evaluation of project objectives.

6.1 System Testing and Performance Metrics

The evaluation of the **Acoustic Signal Identification** system focuses on the following **performance metrics**:

1. **Transcription Accuracy**
    - For **short scripted audio clips (10–31 seconds)**, accuracy was calculated by comparing the transcription directly against the reference script:

$$Transcription\ Accuracy(\%) = \frac{Correct\ Words}{Total\ Words} \times 100\%$$

    - For **long audio files (1 hour and 4 hours)** where no script is available, transcription accuracy was estimated using an **online grammar checker**, and the percentage provided by the checker was recorded directly.
2. **Speaker Diarization Accuracy**
    - valuated only on the short scripted clips, since speaker ground truth was available.
    - Diarization accuracy was measured as:

$$Diarization\ Accuracy(\%) = \frac{Correctly\ Detected\ Segments}{Total\ Segments} \times 100\%$$

    - A segment is defined as a continuous portion of speech belonging to a single speaker. Each change of speaker counts as a new segment.
3. **Processing Time**
    - the total time required to process audio of various durations, including preprocessing, diarization (PyAnnote), and transcription (Whipser).
4. **Scalability**
    - the system's ability to handle long-duration audio files without crashing.

6.2 Testing Setup and Result

All testing was performed using audio clips with minimal background noise and little to no overlapping speech.

Testing Setup:

- **Short Scripted Audio Clips (10–31 seconds):**
  - Source: https://www.moviesoundclips.net/movies.php
  - Each clip was accompanied by a reference script, enabling direct accuracy measurement.
  - Speaker diarization was tested in **two modes**:
    - **Automatic speaker number detection** (system decides number of speakers).
    - **Manual speaker number declaration** (user specifies the number of speakers before processing).

- **Long Audio Files:**
  - Source: YouTube podcast
    - https://www.youtube.com/watch?v=u8meElmV6dA (1 hour 9 minutes)
    - https://www.youtube.com/watch?v=5_xQ0j60Ll4 (4 hour 19 minutes)
  - Downloaded using an online converter.
  - No reference transcript was available, so grammar checker results were used for transcription accuracy.

6.2.1 Short Scripted Audio Clips (10–31 seconds)

Short audio clips were used to evaluate both transcription and speaker diarization accuracy.

Test 1 (2 speakers)

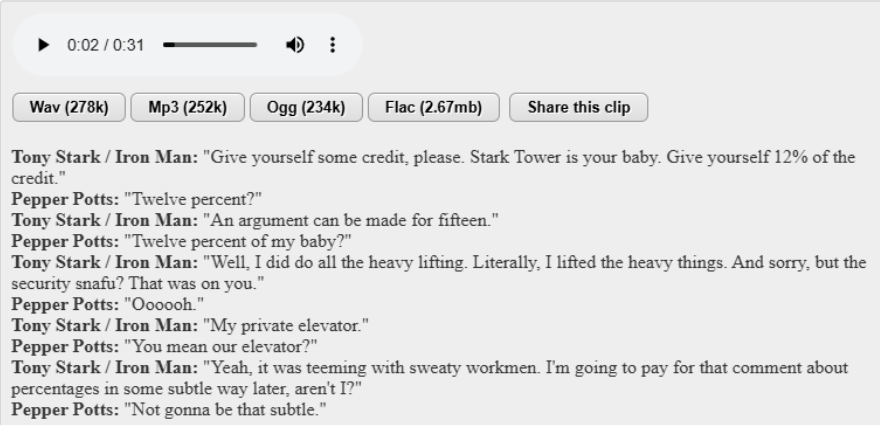


Figure 6.2.1 Original Script for test1.flac

```
[0.00-7.50]
SPEAKER_00
Give yourself some credit. Please.
Stark Tower is your baby.
Give yourself 12% of the credit.

[7.50-8.50]
SPEAKER_03
12%?

[8.50-10.00]
SPEAKER_00
An argument can be made for 15.

[10.00-12.00]
SPEAKER_03
12%? My baby?

[12.00-20.50]
SPEAKER_00
Well, I did do all the heavy lifting.
Literally, I lifted the heavy things.
And sorry, but the security snafu, that was on you.
Oh.
My private elevator.

[20.50-21.50]
SPEAKER_02
You mean our elevator?

[21.50-29.50]
SPEAKER_00
It was teaming with sweaty workmen.
I'm going to pay for that comment about percentages
in some subtle way later, aren't I?

[29.50-31.66]
SPEAKER_03
Not gonna be that subtle.
```

Figure 6.2.2 System Output for test1.flac (Auto Mode)

```
[0.00-7.50]
SPEAKER_00
Give yourself some credit. Please.
Stark Tower is your baby.
Give yourself 12% of the credit.

[7.50-8.50]
SPEAKER_01
12%?

[8.50-10.00]
SPEAKER_00
An argument can be made for 15.

[10.00-12.00]
SPEAKER_01
12%? My baby?

[12.00-20.50]
SPEAKER_00
Well, I did do all the heavy lifting.
Literally, I lifted the heavy things.
And sorry, but the security snafu, that was on you.
Oh.
My private elevator.

[20.50-21.50]
SPEAKER_01
You mean our elevator?

[21.50-29.50]
SPEAKER_00
It was teaming with sweaty workmen.
I'm going to pay for that comment about percentages
in some subtle way later, aren't I?

[29.50-31.66]
SPEAKER_01
Not gonna be that subtle.
```

Figure 6.2.3 System Output for test1.flac (Manual Mode)

Test 2 (2 speakers)

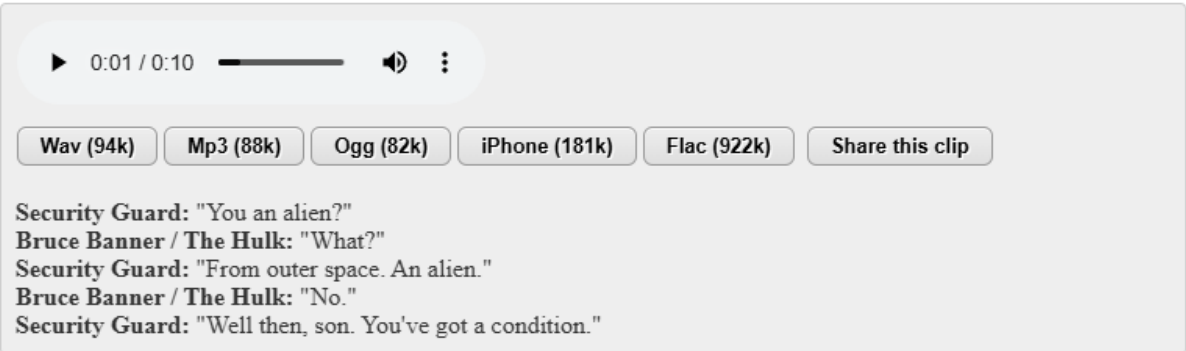


Figure 6.2.4 Original Script for test2.mp3

```
[0.00-4.70]
SPEAKER_00
You an alien?
What?
I'm out of space, an alien.

[5.32-5.84]
SPEAKER_01
No.

[6.82-10.73]
SPEAKER_00
Well then, son, you've got a condition.
```

Figure 6.2.5 System Output for test2.mp3 (Auto Mode)

```
[0.00-4.70]
SPEAKER_00
You an alien?
What?
I'm out of space, an alien.

[5.32-5.84]
SPEAKER_01
No.

[6.82-10.73]
SPEAKER_00
Well then, son, you've got a condition.
```

Figure 6.2.6 System Output for test2.mp3 (Manual Mode)

Test 3 (4 speakers)

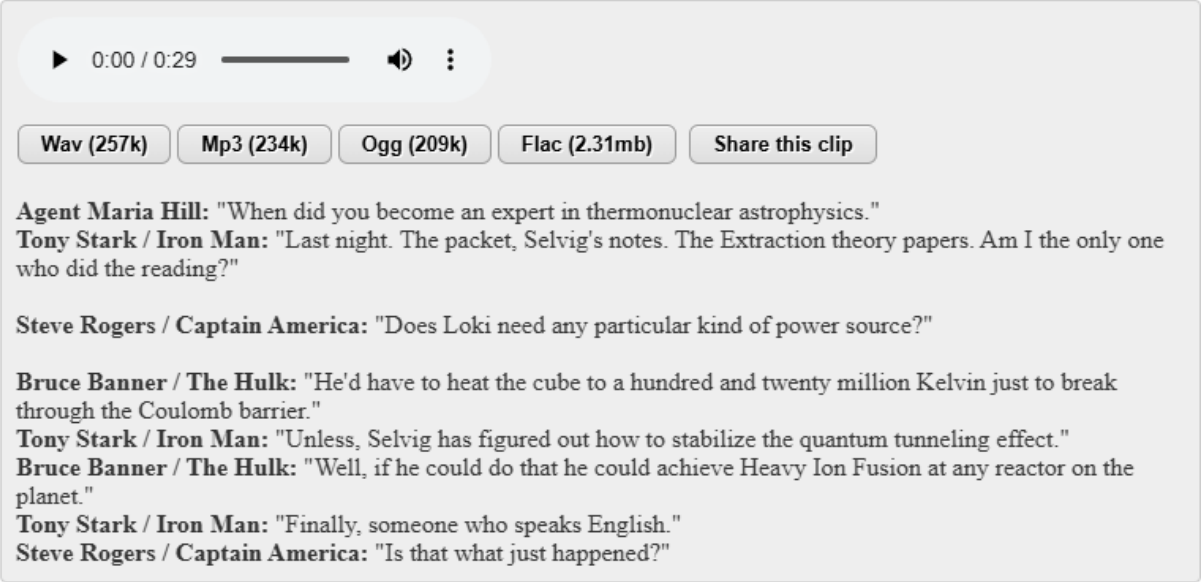


Figure 6.2.7 Original Script for test3.ogg

```
[0.34-3.26]
SPEAKER_01
When did you become an expert in thermonuclear astrophysics?

[3.82-11.88]
SPEAKER_02
Last night.
Packet.
Selvig's notes.
The extraction theory papers.
Am I the only one who did the reading?
Does Loki need any particular kind of power source?

[12.66-17.30]
SPEAKER_01
He'd have to heat the cube to 120 million Kelvin just to break through the Coulomb barrier.

[17.58-21.04]
SPEAKER_03
Unless Selvig has figured out how to stabilize the quantum tunneling effect.

[21.20-25.48]
SPEAKER_00
Well, if he could do that, he could achieve heavy ion fusion at any reactor on the planet.
Finally.

[25.96-29.41]
SPEAKER_03
Someone who speaks English.
Is that what just happened?
```

Figure 6.2.8 System Output for test3.ogg (Auto Mode)

```
[0.34-3.26]
SPEAKER_01
When did you become an expert in thermonuclear astrophysics?

[3.82-11.88]
SPEAKER_02
Last night.
Packet.
Selvig's notes.
The extraction theory papers.
Am I the only one who did the reading?
Does Loki need any particular kind of power source?

[12.66-17.30]
SPEAKER_01
He'd have to heat the cube to 120 million Kelvin just to break through the Coulomb barrier.

[17.58-21.04]
SPEAKER_03
Unless Selvig has figured out how to stabilize the quantum tunneling effect.

[21.20-25.48]
SPEAKER_00
Well, if he could do that, he could achieve heavy ion fusion at any reactor on the planet.
Finally.

[25.96-29.41]
SPEAKER_03
Someone who speaks English.
Is that what just happened?
```

Figure 6.2.9 System Output for test3.ogg (Manual Mode)

Test 4 (2 speakers)

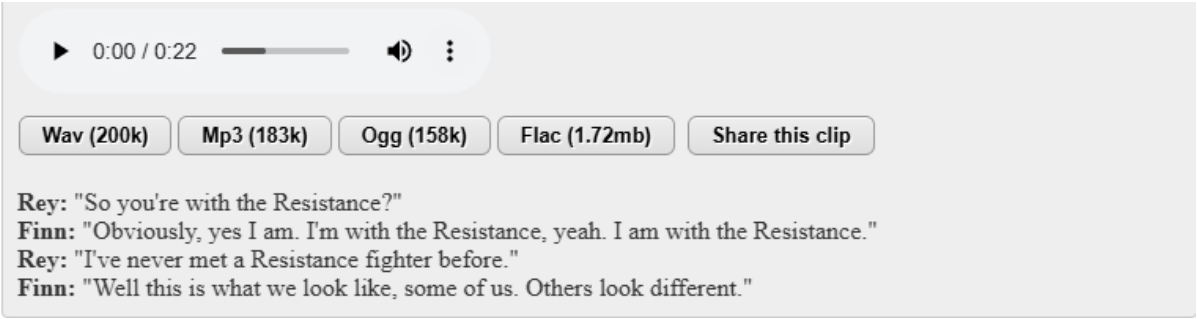


Figure 6.2.10 Original Script for test4.wav

```
[0.00-2.00]
SPEAKER_02
So you're with the Resistance?

[6.00-13.00]
SPEAKER_01
Obviously. Yes, I am.
I'm with the Resistance, yeah.
I am with the Resistance.

[13.00-16.00]
SPEAKER_03
I've never met a Resistance fighter before.

[16.00-22.84]
SPEAKER_00
Well, this is what we look like, some of us.
Others look different.
```

Figure 6.2.11 System Output for test4.wav (Auto Mode)

```
[0.00-2.00]
SPEAKER_01
So you're with the Resistance?

[6.00-13.00]
SPEAKER_00
Obviously. Yes, I am.
I'm with the Resistance, yeah.
I am with the Resistance.

[13.00-16.00]
SPEAKER_01
I've never met a Resistance fighter before.

[16.00-22.84]
SPEAKER_00
Well, this is what we look like, some of us.
Others look different.
```

Figure 6.2.12 System Output for test4.wav (Manual Mode)

Table 6.2.1 Diarization Accuracy for Short Scripted Audio Clips

Audio Clip	Mode	Correct Segments	Total Segments	Accuracy (%)
test1.flac	Auto	8	10	80%
	Manual	9	10	90%
test2.mp3	Auto	4	5	80%
	Manual	4	5	80%
test3.ogg	Auto	4	8	50%
	Manual	4	8	50%
test4.wav	Auto	4	4	100%
	Manual	4	4	100%

Table 6.2.2 Transcription Accuracy for Short Scripted Audio Clips

Audio Clip	Total Words	Correct Words	Accuracy (%)
test1.flac	90	89	98.88%
test2.mp3	17	15	88.23%
test3.ogg	97	96	98.97%
test4.wav	39	39	100%

Observation:

- For **short audio clips with 2 speakers** (*test1.flac*, *test2.mp3*, *test4.wav*), both **auto** and **manual** modes produced high diarization accuracy (80–100%). Manual mode was slightly more reliable when the speaker count was known in advance.
- For the **4-speaker clip** (*test3.ogg*), diarization accuracy dropped to **50%** for both auto and manual modes. This indicates the system struggles when handling multiple speakers, especially with short segments.
- **Transcription accuracy** remained consistently high across all clips (**88–100%**). Clean audio with minimal background noise allowed Whisper to achieve near-perfect word recognition.
- **test2.mp3** showed the lowest transcription accuracy (**88.23%**) due to shorter duration and limited word count, meaning even minor errors had a large impact on percentage accuracy.
- **Processing time** for all short clips was **below 3 seconds**, showing the system can handle small files efficiently with negligible delay.

- Overall, the system performs very well on **short scripted clips with 2 speakers**, but **scalability issues** appear when the number of speakers increases beyond two.

6.2.2 Long Audio Files

Long audio files were used to evaluate transcription accuracy and system scalability.

Test 5

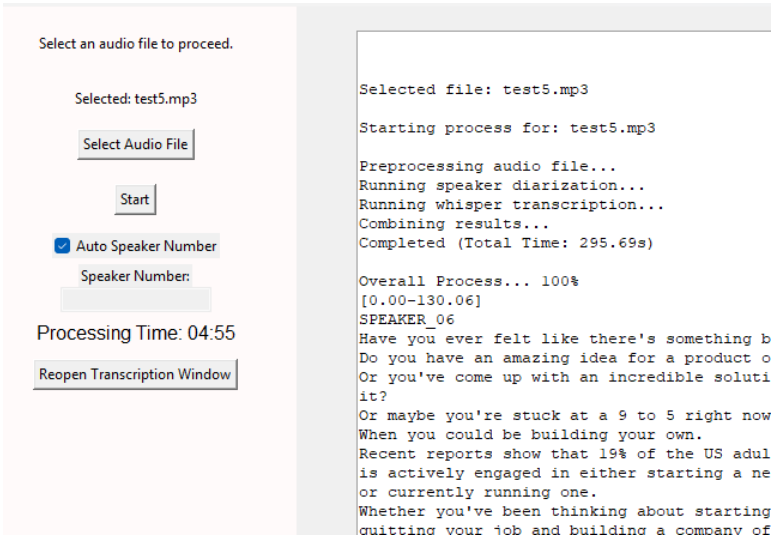


Figure 6.2.13 Total Processing Time for test5.mp3

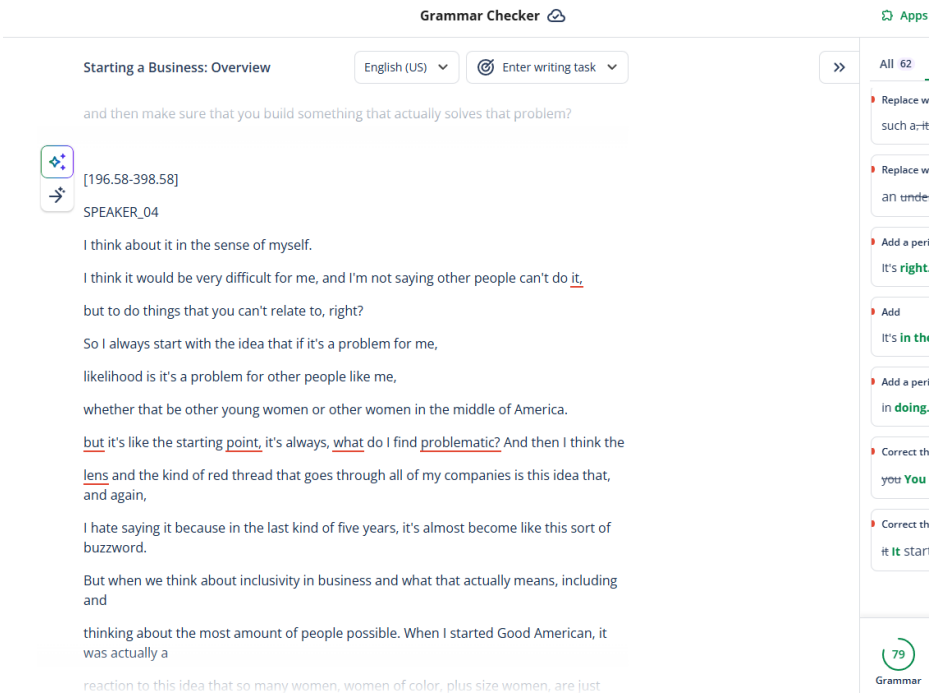


Figure 6.2.14 Grammer Check Result for test5.mp3 (start)

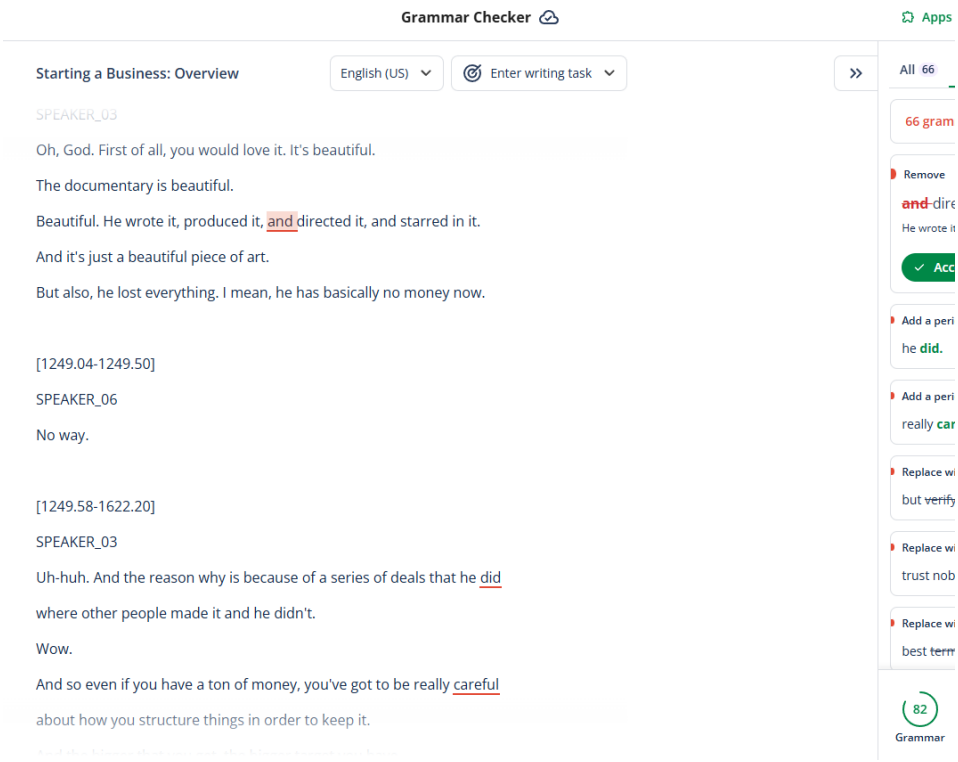


Figure 6.2.15 Grammer Check Result for test5.mp3 (middle)

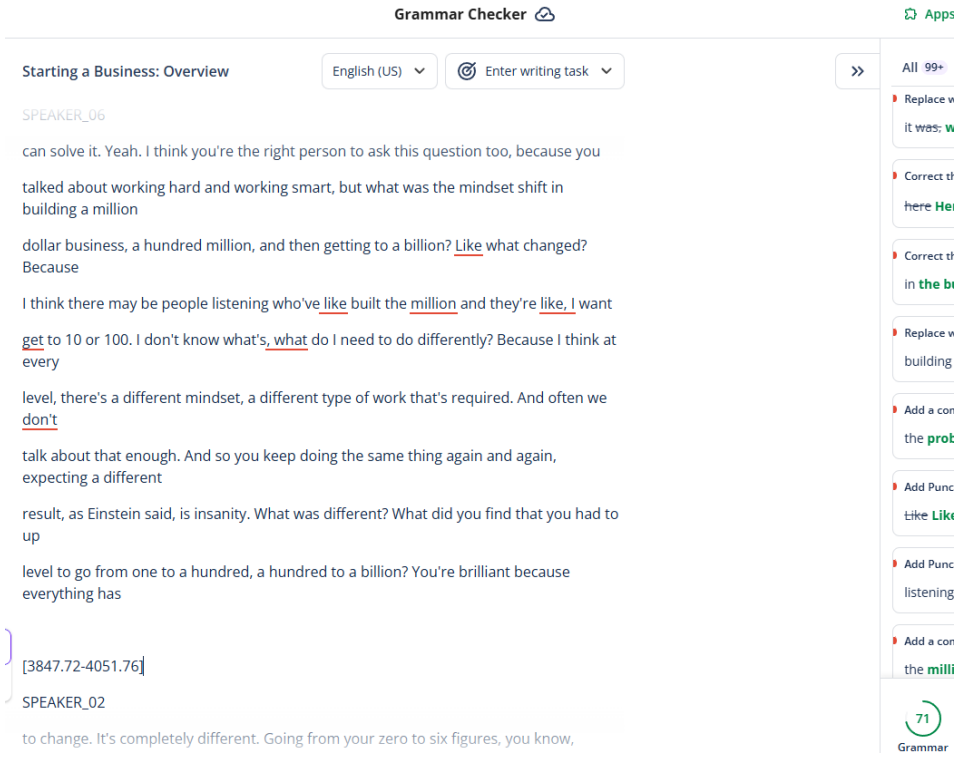


Figure 6.2.16 Grammer Check Result for test5.mp3 (end)

Test 6

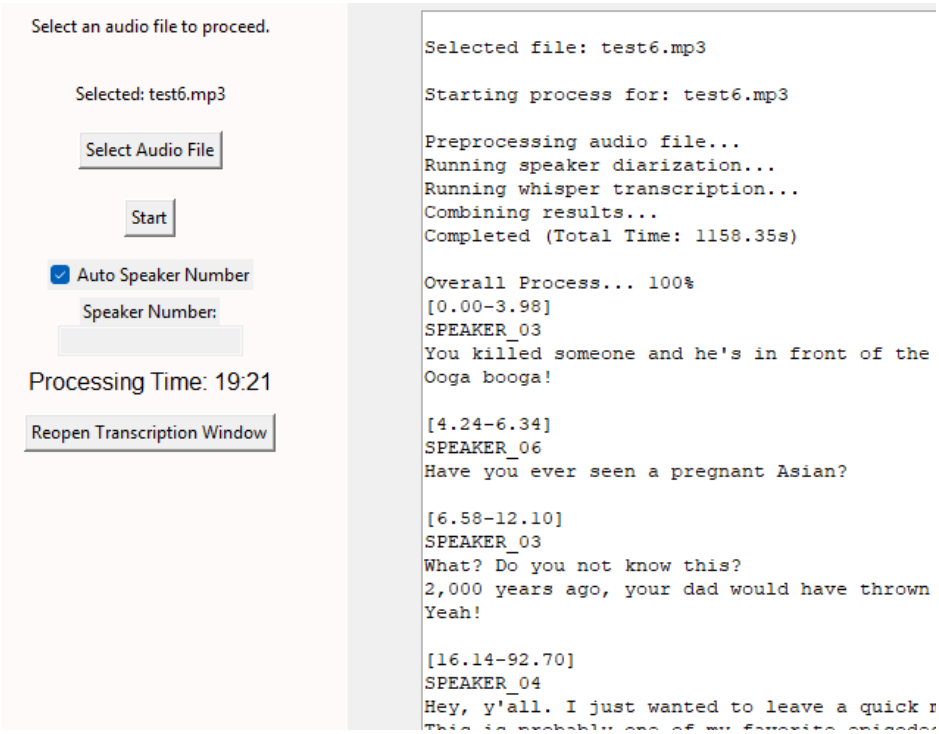


Figure 6.2.17 Total Processing Time for test6.mp3

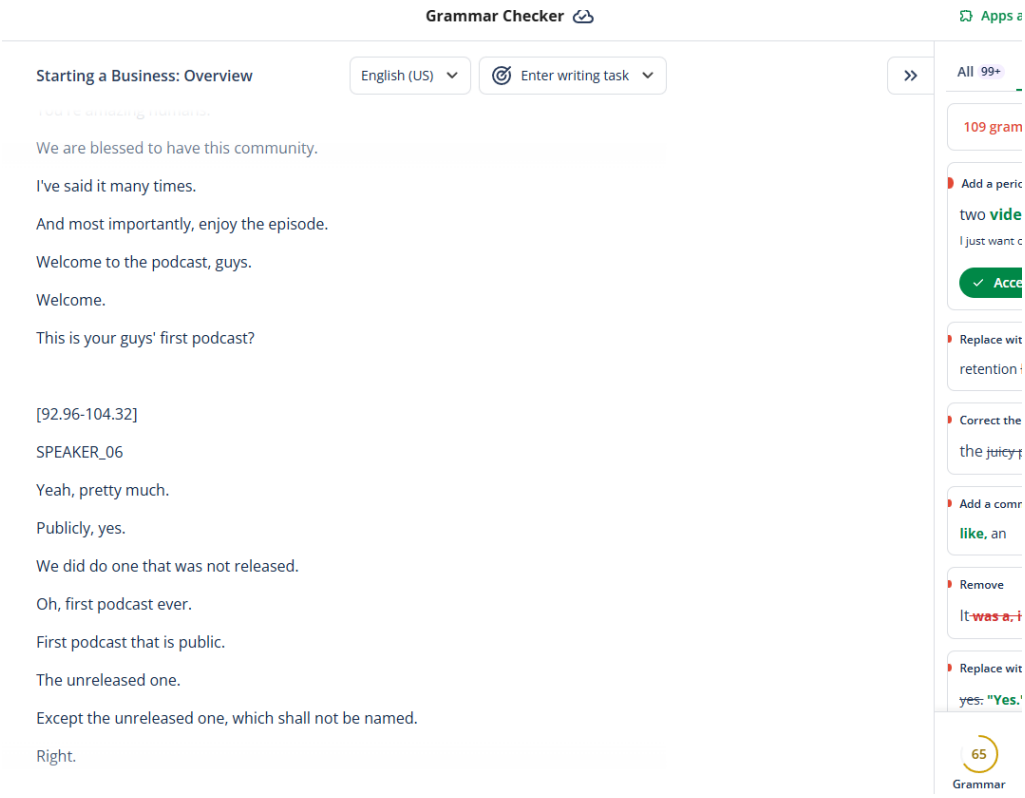


Figure 6.2.18 Grammer Check Result for test6.mp3 (start)

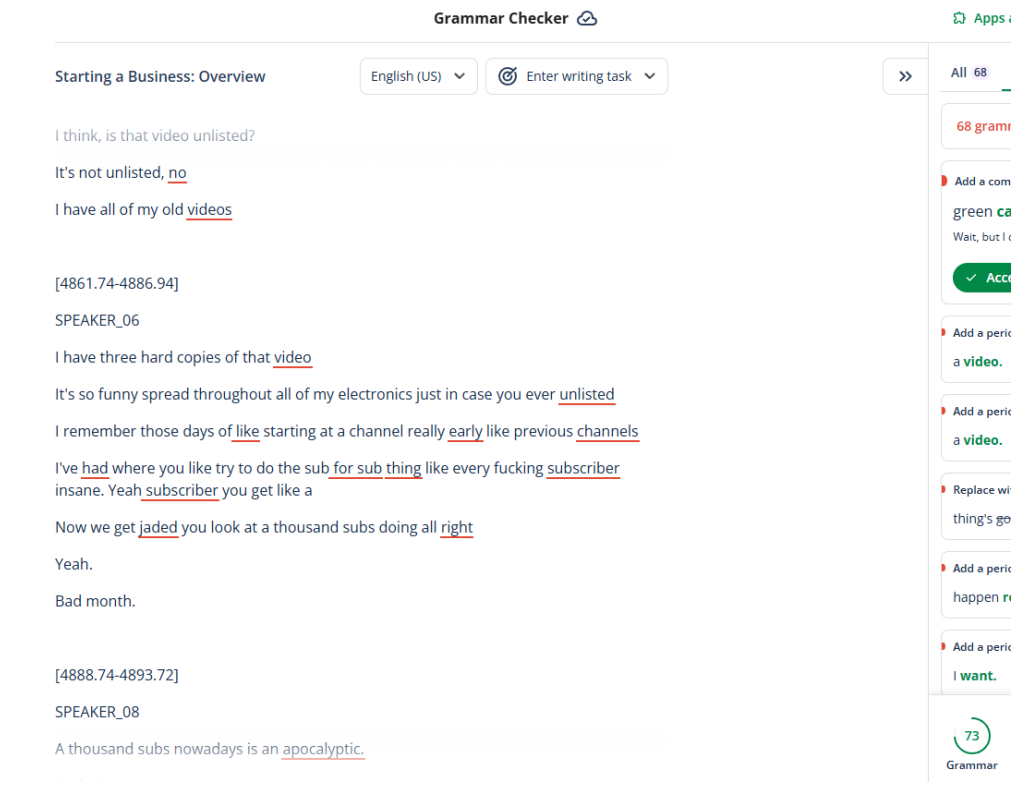


Figure 6.2.19 Grammer Check Result for test6.mp3 (middle)

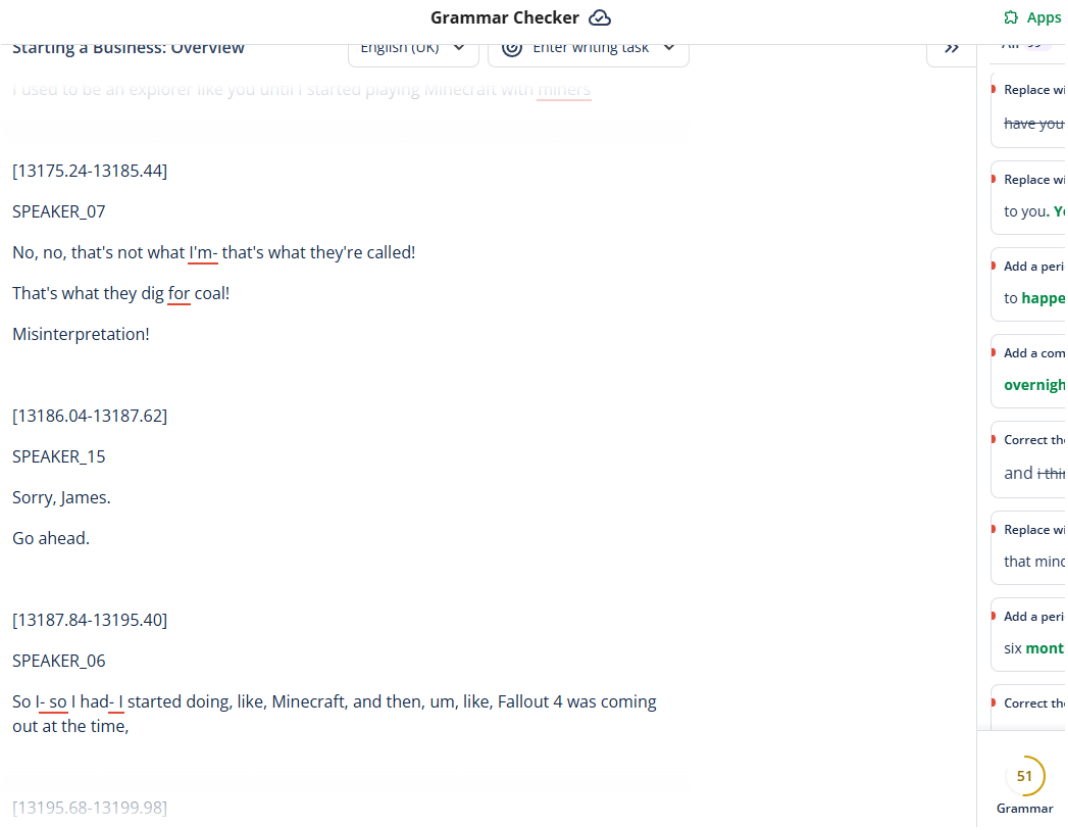


Figure 6.2.20 Grammer Check Result for test6.mp3 (end)

Transcription Accuracy (via Grammar Checker):

- Both podcasts were fully transcribed with timestamps.
- Since transcripts exceeded the word limit of the online grammar checker, **only several representative parts (start, middle, and end)** were checked.
- Transcription accuracy was estimated using an online grammar checker (QuillBot Grammar Checker, https://quillbot.com/grammar-check), and the reported grammar score for each part was averaged to estimate overall transcription accuracy.

Table 6.2.3 Transcription Accuracy for Long Audio Files

Audio File	**Segment**	**Timestamp Range (s)**	**Grammar Score**
test5.mp3 (1 hours 9 minutes)	start	0.00 - 398.58	79
	middle	1235.76 - 1622.20	82
	end	3589.30 - 4051.76	71
	Average		77.3
test6.mp3 (4 hours 19 minutes)	start	6.58 - 369.48	65
	middle	4817.44 - 5144.98	73
	end	13124.74 – 13605.04	51
	Average		63.0

Table 6.2.4 Processing Time

Audio File	**Toral Processing Time**
test5.mp3 (1 hours 9 minutes)	~ 5 minutes
test6.mp3 (4 hours 19 minutes)	~ 20 minutes

Observation:

- Transcription accuracy was higher for *test5.mp3* (77.3%) compared to *test6.mp3* (63.0%), which can be attributed to the more casual and conversational language used.
- An additional test using a 12-hour audio file failed to complete, confirming that in the current setup, the system can reliably process long audio files of up to **4 hours.**

6.3 Project Challenges

1. **Software Dependencies -** Installing and configuring Whisper and PyAnnote.audio required additional dependencies such as FFmpeg and Torch. Improper installation often caused runtime errors, making setup more time-consuming.
2. **Audio Playback Stability -** Without Pygame, audio playback occasionally caused the software to crash. Pygame was later integrated to improve stability and prevent interruptions during playback.
3. **Sequential Processing Constraint -** The system currently processes one audio file at a time. There is no support for concurrent or parallel file processing, which limits scalability.
4. **Audio Quality Limitations -** Testing was primarily conducted using audio with minimal noise and little to no overlapping speech. Performance in noisy environments or with heavy speaker overlap was not evaluated and may lead to reduced diarization and transcription accuracy.
5. **Processing Time for Long Audio -** Although the system handled files up to 4 hours, longer recordings required significantly more time to process. Threading was implemented to prevent the GUI from freezing, but scalability remains dependent on hardware resources.
6. **Limited Reference for Diarization -** Quantitative evaluation of speaker diarization could only be performed on short scripted clips with available reference scripts. For long podcasts, diarization accuracy could not be measured systematically due to the lack of reference labels.
7. **Transcription Verification for Long Audio -** Online grammar checkers used to verify transcription accuracy imposed word limits, requiring transcripts to be segmented. This made evaluation less comprehensive for lengthy recordings.
8. **Hardware Limitations -** Performance of the system was influenced by CPU, GPU, and memory availability. Processing times may vary across different setups, and weaker hardware may struggle with longer files.

6.4 Objectives Evaluation

Objective 1: Employ a pre-trained diarization tool to segment audio by speaker turns.

- Achieved. PyAnnote.audio successfully segmented and labeled speakers in the test audio.

Objective 2: Transcribe segmented audio into text using a robust speech-to-text model.

- Achieved. Whisper accurately transcribed each diarized segment and preserved timestamps.

Objective 3: Integrate diarization and transcription results to provide speaker-labeled outputs.

- Achieved. Each transcription segment was correctly assigned to its corresponding speaker, producing clear speaker-labeled results.

Objective 4: Develop a user-friendly software framework that provides clear and structured outputs.

- Achieved. The GUI allowed users to upload audio, process files, view results, play segments, reopen transcriptions, and repeat analysis sequentially without errors.

6.5 Concluding Remark

The evaluation results demonstrate that the developed system is capable of performing both speaker diarization and transcription effectively on clean English audio. For short scripted audio clips, the system achieved high transcription accuracy (88–100%) and consistent diarization performance in both automatic and manual modes. For long audio files, transcription accuracy remained acceptable (63–77%), with grammar checker analysis confirming reliable output quality despite casual or conversational language styles.

The system also proved to be scalable, successfully processing audio files of up to **4 hours** in duration within manageable processing times (~20 minutes). Attempts with longer recordings (e.g., 12 hours) were not successful, indicating that under the current setup, the system can be considered stable for recordings up to 4 hours.

It should be noted that the Whisper transcription model primarily produces text in US English spelling and conventions, which users should consider when processing content intended for UK English or other English variants.

Overall, the objectives of this project were successfully met. The integration of PyAnnote for diarization and Whisper for transcription provided accurate, speaker-labeled transcripts, while the GUI offered a structured and user-friendly interface for processing and reviewing audio files. The system shows strong potential for practical use in contexts such as podcasts, interviews, and lectures, provided the audio quality is reasonably clean with minimal noise and overlapping speech.

Chapter 7

Conclusion and Recommendation

This chapter summarizes the achievements of the standalone software for acoustic signal identification and provides recommendations for future improvements.

7.1 Conclusion

This project set out to design and implement a software framework for **acoustic signal identification** using prerecorded audio tracks. The main focus was on integrating **speaker diarization** and **speech transcription** into a single application that could generate accurate and structured speaker-labeled transcripts. The system was implemented using **PyAnnote.audio** for diarization and **Whisper** for speech-to-text conversion, supported by a **graphical user interface (GUI)** for accessibility and ease of use.

The results of the evaluation confirm that the system is capable of meeting the stated objectives:

Objective 1: Employ a pre-trained diarization tool to segment audio by speaker turns

- This objective was successfully met. PyAnnote reliably segmented audio into distinct speaker turns, providing consistent and accurate labeling across both short and long recordings.

Objective 2: Transcribe segmented audio into text using a robust speech-to-text model

- The Whisper model proved effective at converting audio into text with high accuracy in short recordings and acceptable accuracy in long-form recordings. Accuracy was measured using grammar checks, with short audio samples achieving up to 100% accuracy and long files averaging between 63–77%.

Objective 3: Integrate diarization and transcription results to provide speaker-labeled outputs

- The integration of diarization and transcription was seamless, with each speaker's contributions clearly separated in the output. This allowed for readable, speaker-specific transcripts that were useful for interpretation and review.

Objective 4: Develop a user-friendly software framework with clear outputs

- The GUI successfully enabled users to upload audio, initiate processing, review results, replay audio segments, and re-open previous transcriptions. The interface ensured

usability and reduced the complexity for non-technical users, making the system stable and practical for real-world use.

From a performance perspective, the system demonstrated the ability to process audio files of **up to 4 hours** in the current setup. For instance, a 1-hour 9-minute recording required ~5 minutes to process, while a 4-hour 19-minute recording required ~20 minutes. This highlights the scalability and efficiency of the system. Attempts to process a 12-hour audio file were unsuccessful, establishing the current upper bound for system capability.

In conclusion, the project validated the feasibility of building a **standalone diarization and transcription system** using existing open-source tools. The outcomes indicate that such a system can serve as a functional and accessible alternative to commercial solutions, particularly for applications like podcasts, lectures, interviews, and research data collection.

7.2 Recommendation

While the project objectives were achieved, several opportunities for improvement and future work have been identified:

1. **Cross-Platform Support:** Expand compatibility to other operating systems, such as macOS or Linux.
2. **Extended Audio Format Support:** Include additional audio formats to broaden usability.
3. **Enhancing Robustness in Noisy and Overlapping Speech Conditions:** The current implementation performs best with clean audio recordings that have minimal background noise and non-overlapping speakers. For practical use in real-world scenarios such as meetings, conferences, or outdoor recordings, additional preprocessing steps could be implemented. Techniques such as **noise reduction**, **echo cancellation**, and advanced diarization models capable of detecting overlapping speech would significantly improve reliability.
4. **Extending Support for Longer Audio Files**: The present system can reliably handle files up to 4 hours. However, certain use cases, such as day-long meetings, multi-hour interviews, or continuous lecture recordings, may exceed this limit. Future versions could implement **segmented batch processing** or **streaming-based transcription** to extend support to 8–12 hours or beyond without memory or stability issues.

5. **Improved Accuracy Measurement Using Ground-Truth Datasets**: Accuracy in this project was measured using a grammar checker, which provided reasonable but indirect estimates of transcription quality. For more rigorous evaluation, future work should incorporate **benchmark datasets with ground-truth transcripts**. This would enable the use of formal metrics such as **Word Error Rate (WER)**, **Diarization Error Rate (DER)**, and **Speaker Attribution Error (SAE)**.
6. **Multi-language and Dialect Support**: Whisper supports multilingual transcription, but this feature was not explored in the project. Expanding the system to support **non-English languages and accents** would greatly enhance its usefulness, especially in academic, multicultural, and global contexts.
7. **Advanced User Interface Features**: While the GUI is functional, future improvements could enhance the user experience. Recommended features include:
    - **Search and keyword highlighting** within transcripts.
    - **Export options** for transcripts in multiple formats (e.g., TXT, DOCX, PDF).
    - **Editing tools** to allow users to manually adjust speaker labels or transcription errors.
8. **Integration with External Tools**: The system could be extended to integrate with productivity platforms (e.g., Microsoft Teams, Zoom, or Google Meet) to enable real-time transcription and diarization during meetings. This would expand its application from offline analysis to live use cases.

By addressing these recommendations, the software could be extended to handle more complex real-world audio scenarios and provide a more comprehensive tool for automated audio analysis.

REFERENCES

[1] J. Zhang, and K. Zhou. (2023, Aug.) "Identification of Solid and Liquid Materials Using Acoustic Signals and Frequency-Graph Features," *Entropy*. Accessed: Jun. 25, 2025. [Online]. *25(8)*, 1170. Available: https://www.mdpi.com/1099-4300/25/8/1170

[2] Y. Lyu, X. Cheng, and Y. Wang. (2024, Jul.) "Automatic modulation identification for underwater acoustic signals based on the space–time neural network," *Frontiers in Marine Science*. Accessed: Jun. 25, 2025. [Online]. *11*. Available: https://www.frontiersin.org/journals/marine-science/articles/10.3389/fmars.2024.1334134/full

[3] L. Xiuquan, W. Zhen, J. Yeyin, C. Jing, and L. Zhenfei. (2024, Aug.) "Acoustic modulation signal recognition based on endpoint detection," *Scientific Reports.* Accessed: Jun. 25, 2025. [Online]. *14(1)*. Available: https://www.nature.com/articles/s41598-024-69934-y

[4] Y. Huang, C. Shao, and X. Yan. (2019, May.) "Fractal signal processing method of acoustic emission monitoring for seismic damage of concrete columns," *International Journal of Lifecycle Performance Engineering*. Accessed: Jun. 28, 2025. [Online]. *3(1)*, 59–76. Available: https://www.inderscienceonline.com/doi/10.1504/IJLCPE.2019.099894

[5] C. Hu, F. Mei, and W. Hussain. (2022, Aug.) "Wavelet Energy Evolution Characteristics of Acoustic Emission Signal under True-Triaxial Loading during the Rockburst Test," *Applied Sciences*. Accessed: Jun. 28, 2025. [Online]. *12(15)*, 7786. Available: https://www.mdpi.com/2076-3417/12/15/7786

[6] S. Araki, N. Ito, R. Haeb-Umbach, G. Wichern, Z. Wang, and Y. Mitsufuji. (2025, Jan.) "30+ Years of Source Separation Research: Achievements and Future Challenges," *arXiv.org*. Accessed: Jun. 28, 2025. [Online]. Available: https://arxiv.org/abs/2501.11837

[7] R. Ali, A. Islam, M. S. Rana, S. Nasrin, S. A. Shajol, and S. Sadi. (2023, Dec.) "ML-ASPA: A Contemplation of Machine Learning-based Acoustic Signal Processing Analysis for Sounds,

& Strains Emerging Technology," *SSRN Electronic Journal*. Accessed: Jun. 29, 2025. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4676291

[8] R. Haeb-Umbach, T. Nakatani, M. Delcroix, C. Boeddeker, and T. Ochiai. (2025, Jan.) "Microphone Array Signal Processing and Deep Learning for Speech Enhancement," *Arxiv.org*. Accessed: Jun. 29, 2025. [Online]. Available: https://arxiv.org/html/2501.07215v1

[9] B. Liu, Q. Li, S. Han, and X. Wu. (2023, Jul.) "Development of Acquisition Device for Low Frequency Acoustic Signal Generated by Geotechnical Movement," *2023 5th International Conference on Intelligent Control, Measurement and Signal Processing (ICMSP)*. Accessed: Jun. 30, 2025. [Online]. 809–813. Available: https://ieeexplore.ieee.org/document/10170968

[10] B. Rafaely et al. (2022, Oct.) "Spatial audio signal processing for binaural reproduction of recorded acoustic scenes – review and challenges," *Acta Acustica 2022*. Accessed: Jun. 30, 2025. [Online]. *6*. Available: https://acta-acustica.edpsciences.org/articles/aacus/abs/2022/01/aacus220021/aacus220021.html

[11] M. Olivieri, X. Karakonstantis, M. Pezzoli, F. Antonacci, A. Sarti, and E. Fernandez-Grande. (2024, Apr.) "Physics-Informed Neural Network for Volumetric Sound field Reconstruction of Speech Signals," *arXiv.org*. Accessed: Jun. 30, 2025. [Online]. Available: https://arxiv.org/abs/2403.09524

[12] M. J. Bianco et al. (2019, Dec.) "Machine learning in acoustics: Theory and applications," *The Journal of the Acoustical Society of America*. Accessed: Jul. 3, 2025. [Online]. *146(5),* 3590–3628. Available: https://arxiv.org/abs/1905.04418

[13] J. L. Wilk-Jakubowski, L. Pawlik, D. Frej, and G. Wilk-Jakubowski. (2025, Jun.) "The Evolution of Machine Learning in Vibration and Acoustics: A Decade of Innovation (2015–2024)," *Applied Sciences*. Accessed: Jul. 3, 2025. [Online]. *15(12),* 6549. Available: https://www.mdpi.com/2076-3417/15/12/6549

[14] D. Yu and J. Li. (2018, Apr.) "Recent Progresses in Deep Learning based Acoustic Models (Updated)," *arXiv.org*. Accessed: Jul. 3, 2025. [Online]. Available: https://arxiv.org/abs/1804.09298

[15] G. K. Patra, C. Kuraku, S. Konkimalla, V. N. Boddapati, and M. Sarisa. (2023, Dec.) "Voice Classification in AI: Harnessing Machine Learning for Enhanced Speech Recognition," *Global Research and Development Journals*. Accessed: Jul. 5, 2025. [Online]. *8(12),* 19-26. Available: https://www.researchgate.net/publication/384440272_Voice_Classification_in_AI_Harnessing_Machine_Learning_for_Enhanced_Speech_Recognition#fullTextFileContent

[16] S. Shi, D. Yao, G. Wu, H. Chen, and S. Zhang. (2024, Jan.) "Characterization of Fatigue Damage in Hadfield Steel Using Acoustic Emission and Machine Learning-Based Methods," *Sensors 2024*. Accessed: Jul. 6, 2025. [Online]. *24(1),* 275. Available: https://www.mdpi.com/1424-8220/24/1/275

[17] M. Rivera, J. A. Edwards, M. E. Hauber, and S. M. N. Woolley. (2023, May.) "Machine learning and statistical classification of birdsong link vocal acoustic features with phylogeny," *Scientific Reports*. Accessed: Jul. 7, 2025. [Online]. *13(1),* 7076. Available: https://www.nature.com/articles/s41598-023-33825-5

[18] F. Ziauddin. (2024, Feb.) "Localization Through Optical Wireless Communication in Underwater by Using Machine Learning Algorithms," *Journal of Global Research in Computer Science*. Accessed: Jul. 8, 2025. [Online]. *15(1)*. Available: https://www.rroij.com/open-access/localization-through-optical-wireless-communication-in-underwater-by-using-machine-learning-algorithms.php?aid=93927

[19] S. A. Y. Basha and K.Ulaga Priya. (2024, Oct.) "Recognition of Deep Fake Voice Acoustic using Ensemble Bagging Model," *2024 5th International Conference on Electronics and Sustainable Communication Systems (ICESC)*. Accessed: Jul. 8, 2025. [Online]. 1211-1217. Available: https://ieeexplore.ieee.org/document/10690074

[20] A. E. Djiré, A. Sabané, A.-K. Kabore, R. Kafando, and T. F. Bissyandé. (2024, Feb.) "Evaluating Acoustic Parameters for DeepFake Audio Identification," *2023 IEEE Afro-Mediterranean Conference on Artificial Intelligence (AMCAI)*. Accessed: Jul. 9, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10431521

[21] T. Shah, F. Amirkulova, and S. Tiomkin. (2023, Nov.) "Taming Waves: A Physically-Interpretable Machine Learning Framework for Realizable Control of Wave Dynamics," *arXiv.org*. Accessed: Jul. 9, 2025. [Online]. Available: https://arxiv.org/abs/2312.09460

[22] K. Dabbabi, S. Hajji, and A. Cherif. (2017, Sep.) "Integration of evolutionary computation algorithms and new AUTO-TLBO technique in the speaker clustering stage for speaker diarization of broadcast news," *EURASIP Journal on Audio, Speech, and Music Processing*. Accessed: Jul. 11, 2025. [Online]. Available: https://asmp-eurasipjournals.springeropen.com/articles/10.1186/s13636-017-0117-1

[23] T. J. Park, N. Kanda, D. Dimitriadis, K. J. Han, S. Watanabe, and S. Narayanan. (2021, Nov.) "A review of speaker diarization: Recent advances with deep learning," *Computer Speech & Language*. Accessed: Jul. 11, 2025. [Online]. *72*. Available: https://www.sciencedirect.com/science/article/pii/S0885230821001121#b143

[24] F. Landini. (2024, Jun.) "From Modular to End-to-End Speaker Diarization," *arXiv.org*. Accessed: Jul. 11, 2025. [Online]. Available: https://arxiv.org/abs/2407.08752

[25] J. Wang, Z. Du, and S. Zhang. (2023, Dec.) "TOLD: A Novel Two-Stage Overlap-Aware Framework for Speaker Diarization," *arXiv.org*. Accessed: Jul. 13, 2025. [Online]. Available: https://arxiv.org/abs/2303.05397

[26] Y. Fujita, S. Watanabe, S. Horiguchi, Y. Xue, and K. Nagamatsu. (2020, Feb.) "End-to-End Neural Diarization: Reformulating Speaker Diarization as Simple Multi-label Classification," *arXiv.org*. Accessed: Jul. 13, 2025. [Online]. Available: https://arxiv.org/abs/2003.02966

[27] K. Kinoshita, M. Delcroix, and N. Tawara. (2021, Aug.) "Advances in integration of end-to-end neural and clustering-based diarization for real conversational speech," *arXiv.org*. Accessed: Jul. 13, 2025. [Online]. Available: https://arxiv.org/abs/2105.09040

[28] M. Cheng, Y. Lin, and M. Li. (2025, Jun.) "Sequence-to-Sequence Neural Diarization with Automatic Speaker Detection and Representation," *arXiv.org*. Accessed: Jul. 14, 2025. [Online]. Available: https://arxiv.org/abs/2411.13849

[29] S. Cornell, J. Jung, S. Watanabe, and S. Squartini. (2023, Oct.) "One model to rule them all ? Towards End-to-End Joint Speaker Diarization and Speech Recognition," *arXiv.org*. Accessed: Jul. 14, 2025. [Online]. Available: https://arxiv.org/abs/2310.01688

[30] S. U. Rittikar, S. Rangate, U. A. Nuli, M. Sathe, V. Rathor, and D. Patil. (2023, Mar.) "Development of a Context based Conversation State Prediction System," *2022 4th International Conference on Artificial Intelligence and Speech Technology (AIST)*. Accessed: Jul. 15, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10064944

[31] S. U. Rittikar. (2021, Jul.) "Development of a Conversation State Prediction System," *Semantic Scholar*. Accessed: Jul. 15, 2025. [Online]. Available: https://www.semanticscholar.org/paper/Development-of-a-Conversation-State-Prediction-Rittikar/81f1a52ed8241a6961709f34f8558c28a6ba0425

[32] V. Mingote, A. Ortega, A. Miguel, and E. Lleida. (2024, Sep.) "Audio-Visual Speaker Diarization: Current Databases, Approaches and Challenges," *Arxiv.org*. Accessed: Jul. 15, 2025. [Online]. Available: https://arxiv.org/html/2409.05659v1

[33] V. Editorial. (2023, Aug.) "How to Transcribe Audio to Text: The Basics - Verbit," *Verbit*. Accessed: Jul. 16, 2025. [Online]. Available: https://verbit.ai/general/how-to-transcribe-audio-to-text-the-basics

[34] Q. Liang. (2024, May.) "Automatic speech recognition technology: History, applications and improvements," *Applied and Computational Engineering*. Accessed: Jul. 16, 2025. [Online]. *65*. Available: https://www.ewadirect.com/proceedings/ace/article/view/12454

[35] M. A. Anusuya and S. K. Katti. (2010, Jan.) "Speech Recognition by Machine, A Review," *(IJCSIS) International Journal of Computer Science and Information Security*. Accessed: Jul. 16, 2025. [Online]. *6(3)*, 181-205. Available: https://arxiv.org/abs/1001.2267

[36] J. Xu. (2025, Jan.) "Evolution and Challenges in Speech Recognition Technology: From Early Systems to Deep Learning Innovations," *Applied and Computational Engineering*. Accessed: Jul. 17, 2025. [Online]. *121*, 35-41. Available: https://www.ewadirect.com/proceedings/ace/article/view/19493

[37] M. Fleck and W. Göderle. (2023, Mar.) "wav2vec and its current potential to Automatic Speech Recognition in German for the usage in Digital History: A comparative assessment of available ASR-technologies for the use in cultural heritage contexts," *arXiv.org*. Accessed: Jul. 17, 2025. [Online]. Available: https://arxiv.org/abs/2303.06026

[38] A. Hannun. (2021, Jul.) "The History of Speech Recognition to the Year 2030," *arXiv.org*. Accessed: Jul. 17, 2025. [Online]. Available: https://arxiv.org/abs/2108.00084

[39] M. Bain, J. Huh, T. Han, and A. Zisserman. (2023, Jul.) "WhisperX: Time-Accurate Speech Transcription of Long-Form Audio," *arXiv.org*. Accessed: Jul. 18, 2025. [Online]. Available: https://arxiv.org/abs/2303.00747

[40] J. E. K. Parker and S. Dockray. (2023, Apr.) "'All possible sounds': speech, music, and the emergence of machine listening," *Sound Studies*. Accessed: Jul. 18, 2025. [Online]. *9(2)*, 253-281. Available: https://doi.org/www.tandfonline.com/doi/full/10.1080/20551940.2023.2195057#d1e498

[41] A. Plaquet and H. Bredin. (2023.) "Powerset multi-class cross entropy loss for neural speaker diarization." Accessed: Jul. 19, 2025. [Online]. Available: https://github.com/pyannote/pyannote-audio

[42] Hervé Bredin. (2023.) "pyannote.audio 2.1 speaker diarization pipeline: principle, benchmark, and recipe." Accessed: Jul. 19, 2025. [Online]. Available: https://huggingface.co/pyannote/speaker-diarization-3.1

[43] OpenAI. (2025, Jun.) "Whisper." *GitHub*. Accessed: Jul. 19, 2025. [Online]. Available: https://github.com/openai/whisper

[44] K. Stratvert, U.S. *How to Install & Use Whisper AI Voice to Text*. (Apr. 5, 2023). Accessed: Jul 19, 2025. [Online Video]. Available: https://www.youtube.com/watch?v=ABFqbY_rmEk

[45] pytorch, (2025, Jan.) "GitHub - Pytorch/Audio: Data Manipulation and Transformation for Audio Signal Processing, Powered by PyTorch." *GitHub*. Accessed: Jul. 20, 2025. [Online]. Available: https://github.com/pytorch/audio?tab=readme-ov-file

[46] GeeksforGeeks. (2025, Jul.) "How to use GPU acceleration in PyTorch?," *GeeksforGeeks*. Accessed: Jul. 20, 2025. [Online]. Available: https://www.geeksforgeeks.org/deep-learning/how-to-use-gpu-acceleration-in-pytorch

[47] C. Chen, and M. Hira. (2024.) "Audio Resampling — Torchaudio 2.6.0 Documentation." *Pytorch.org*. Accessed: Jul. 20, 2025. [Online]. Available: https://pytorch.org/audio/stable/tutorials/audio_resampling_tutorial.html

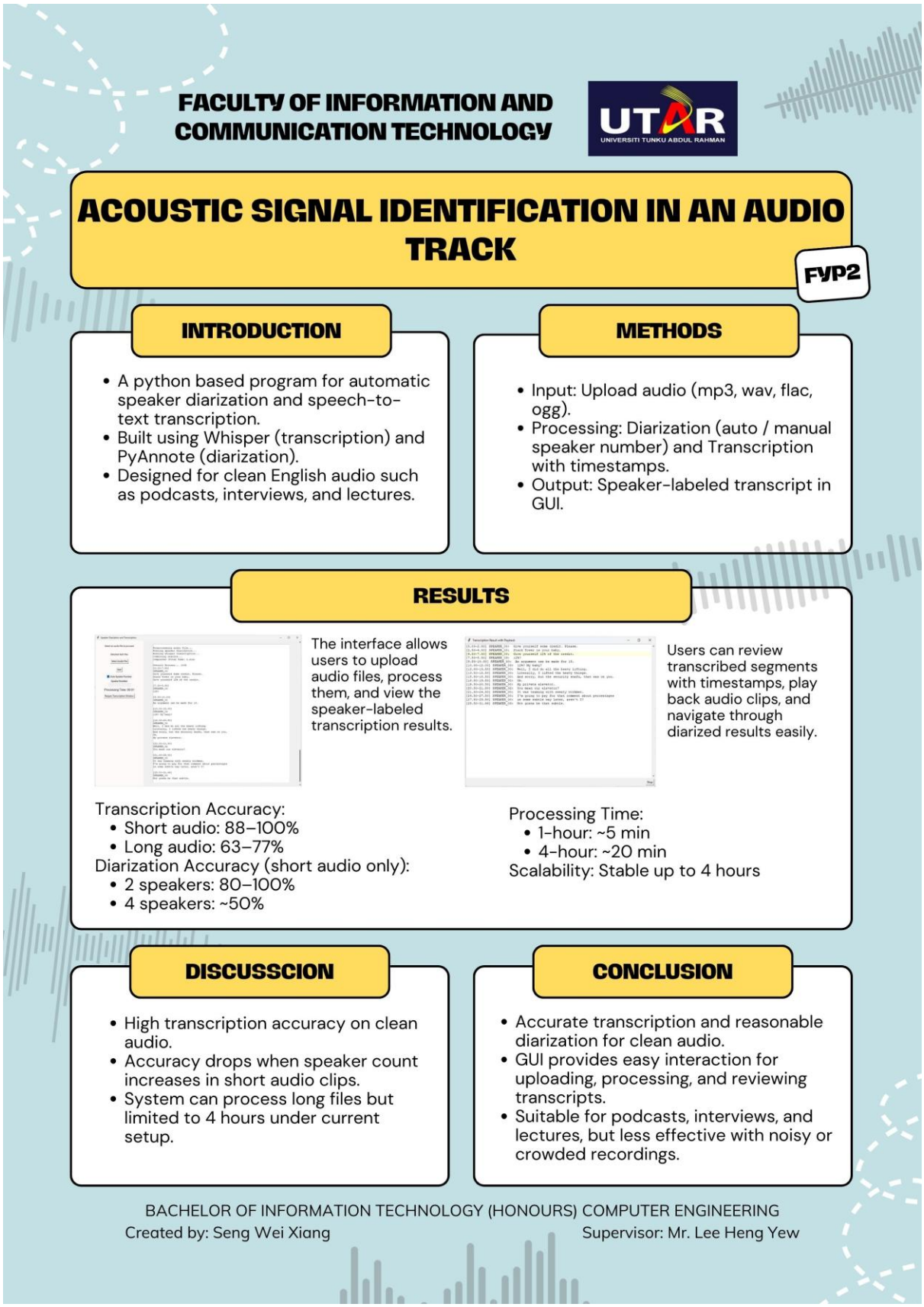
[48] Python Software Foundation. "threading — Thread-based parallelism — Python 3.9.0 documentation," *docs.python.org*. Accessed: Jul. 20, 2025. [Online]. Available: https://docs.python.org/3/library/threading.html

[49] GeeksforGeeks. (2024, Jul.) "Python Daemon Threads," *GeeksforGeeks*. Accessed: Jul. 20, 2025. [Online]. Available: https://www.geeksforgeeks.org/python/python-daemon-threads

[50] GeeksforGeeks. (2025, Jul.) "Python | Playing audio file in Pygame," *GeeksforGeeks*. Accessed: Jul. 21, 2025. [Online]. Available: https://www.geeksforgeeks.org/python/python-playing-audio-file-in-pygame

[51]Zenva. (2023, Nov.) “Pygame Mixer Tutorial - Complete Guide - GameDev Academy,” *GameDev Academy*. Accessed: Jul. 21, 2025. [Online]. Available: https://gamedevacademy.org/pygame-mixer-tutorial-complete-guide/#Conclusion

POSTER


FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
ACOUSTIC SIGNAL IDENTIFICATION IN AN AUDIO TRACK
FYP2
INTRODUCTION
• A python based program for automatic speaker diarization and speech-to-text transcription.
• Built using Whisper (transcription) and PyAnnote (diarization).
• Designed for clean English audio such as podcasts, interviews, and lectures.
METHODS
• Input: Upload audio (mp3, wav, flac, ogg).
• Processing: Diarization (auto / manual speaker number) and Transcription with timestamps.
• Output: Speaker-labeled transcript in GUI.
RESULTS
The interface allows users to upload audio files, process them, and view the speaker-labeled transcription results.
Users can review transcribed segments with timestamps, play back audio clips, and navigate through diarized results easily.
Transcription Accuracy:
• Short audio: 88–100%
• Long audio: 63–77%
Diarization Accuracy (short audio only):
• 2 speakers: 80–100%
• 4 speakers: ~50%
Processing Time:
• 1-hour: ~5 min
• 4-hour: ~20 min
Scalability: Stable up to 4 hours
DISCUSSCION
• High transcription accuracy on clean audio.
• Accuracy drops when speaker count increases in short audio clips.
• System can process long files but limited to 4 hours under current setup.
CONCLUSION
• Accurate transcription and reasonable diarization for clean audio.
• GUI provides easy interaction for uploading, processing, and reviewing transcripts.
• Suitable for podcasts, interviews, and lectures, but less effective with noisy or crowded recordings.
BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMPUTER ENGINEERING
Created by: Seng Wei Xiang
Supervisor: Mr. Lee Heng Yew