An AI-Based System for Pronunciation Assessment and Tracking of Speech Fluency Variations

A thesis submitted in the partial fulfillment for the degree of Bachelor of Technology

Authors:

Akash Paila CS21B1017

Supervised by:

Dr. Dubacharla Gyaneshwar

Indian Institute of Information Technology Raichur

Computer Science and Engineering Department

Raichur-584105

Academic Year: 2021-2025

Declaration

I undersigned hereby declare that the project report **An AI-Based System for Pronunciation Assessment and Tracking of Speech Fluency Variations** submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology of the **Indian Institute of Information Technology Raichur**, is a bonafide work done by me under supervision of **Dr Dubacharla Gyaneshwar.** This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University

Place :

Signature of student :

Date : May 2, 2025

Name of student : Akash Paila

CERTIFICATE

This is to certify that the report entitled **An AI-Based System for Pronunciation Assessment and Tracking of Speech Fluency Variations** submitted by **Akash Paila** to the Indian Institute of Information Technology, Raichur in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science & Engineering is a bonafide record of the project work carried out by him under my guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Project Mentor

Name : Dr Dubacharla Gyaneshwar

Signature :

Acknowledgment

I would like to express my sincere gratitude to who all helped me prepare this Project Report titled **An AI-Based System for Pronunciation Assessment and Tracking of Speech Fluency Variations** and present it satisfactorily. I am especially thankful to my mentor , my guide and my supervisor, **Dr.Dubacharla Gyaneshwar**,in the Department of Computer Science and Engineering for their invaluable guidance, encouragement, and support throughout the course of this research and Implementation

I am also thankful to the faculty and staff of CSE Department, Indian Institute of Information Technology, Raichur for providing a conductive environment and necessary resources for my work. I extend my heartfelt thanks to my family and friends for their constant encouragement, understanding and unwavering support. Finally, I acknowledge all those whose contributions directly or indirectly, have helped me in the successful completion of this thesis.

Abstract

Pronunciation assessment is crucial for non-native speakers learning a new language. Beyond traditional communication tasks, speech-driven electronic devices powered by AI are increasingly pervasive, presenting both opportunities and challenges for accurate speech recognition and analysis. We propose an artificial intelligence (AI) based pronunciation assessment system that records, transcribes, and evaluates user utterances using speech quality indicators such as word error rate (WER) , words per second (WPS),character error rate (CER) and pace identification. The system employs a two-tier architecture: Tier 1 handles transcription using an ASR engine,while Tier 2 applies post-processing to correct domain-specific errors and deliver scores. Experimental results show a 22 % reduction in WER compared to existing ASR systems, along with improved indicators for fluency assessment. This innovative approach holds promise as a self-learning tool to identify and track speech variations, enhancing speaking and reading skills.

Keywords : Automatic Speech Recognition, Pronunciation Analysis, Word Error Rate, Speech Processing, Artificial intelligence.

Contents

List of Figures

List of Tables

List of Acronyms

- **ASR:** Automatic Speech Recognition
- **AI:** Artificial Intelligence.
- **CER:** Character Error Rate.
- **GOP:** Goodness of Pronunciation
- **G-API:** Google Web Speech API.
- **HMM:** Hidden Markov Model.
- **NLP:** Natural Language Processing.
- **ReGex:** Regular Expression.
- **STT:** Speech to Text.
- **TTS:** Text to Speech.
- **WER:** Word Error Rate
- **WPM:** Words Per Minute.
- **WPS:** Words Per Second.

Introduction

Good communication is based on fluency of speech and pronunciation, particularly for non-native speakers learning a new language.At the beginning of learning a new language it is very hard. Social and working relationships are influenced by proper and consistent pronunciation, so it's essential for language learners. It is also essential for language learners due to the increasing interactions.

Also now a days with the growing technology and integration of speech-driven electronic devices powered by Artificial Intelligence (AI), speech analysis has gained significance beyond traditional communication tasks. These technologies prevalent in virtual assistants, language learning tools and voice activated devices, increasingly use Natural Language Processing (NLP) for efficient, real-time processing.

Existing automatic speech recognition (ASR) systems, such as Google Speech-to-Text API, OpenAI Whisper, and Vosk, have achieved great strides in the area of speech transcription. Such systems exhibit high accuracy across general-purpose speech recognition and find extensive usage in a wide range of applications. Their performance, however, suffers with the processing of domain-specific mistakes, pace variability, and inconsistency in fluency. Such limitations point toward the necessity for a stronger architecture that encompasses transcription with targeted error correction and fluency analysis, providing accurate judgment and feedback for non-native speakers.

In order to address these concerns, this study presents a new AI-based automated pronunciation assessment system that detects and tracks speech variation and inaccuracy. The system consists of a two-tier system: tier 1 employs an ASR engine to transcribe speech, and tier 2 applies a post-processing module to fix errors specific to the domain, while also computing critical fluency measures such as word error rate (WER), character error rate (CER), pacing, and words per second (WPS). The course of study integrates the two tiers in order to reduce transcription errors and provide a more holistic measure of fluency. Results demonstrated that the WER is lower than existing ASRs, and the metrics for identifying and evaluating

pacing were better compared to existing ASRs. Based on the innovative use of AI as well as the specific use of post-processing post transcription, this AI based system demonstrates promising potential as a self-learning tool for oral speaking fluency and reading, to be used in a variety of contexts.

The core component of the system is Grammar Analysis, which allows for the automatic detection and correction of grammatical errors in user input, thereby providing real-time feedback to language skill learners. Besides this, the use of Speech-to-Text (STT) technology allows the system to read user input from speech and transform it into written form for analysis. On the other hand, the Text-to-Speech (TTS) module generates natural speech from written content, which helps learners to improve their pronunciation and auditory skills.

Furthermore, the system incorporates Accent-Based Learning capabilities, which tailor the learning experience according to the user's regional or native accent. This ensures that the speech recognition and synthesis modules are adaptive, inclusive, and effective across a diverse user base. By combining these components, the project aims to provide an interactive and personalized language learning experience that bridges the gap between text and speech in real-time educational contexts.

Chapter 1

System Architecture and Methodology

This chapter presents the architectural design and methodological framework adopted in the development of the proposed system. The system is composed of multiple modules working in unison to deliver a comprehensive speech-based English learning and transcription tool. The core components include Speech-to-Text (STT), Text-to-Speech (TTS), grammar analysis, accent detection, voice-based authentication using MFCCs, performance tracking, and YouTube video transcription and summarization.

1.1 System Overview

The system architecture is modular, where each component functions independently and contributes to the overall pipeline. The main modules are:

- **Speech-to-Text (STT):** Converts spoken input to text using Google's Speech Recognition API.
- **Text-to-Speech (TTS):** Converts written content into spoken output using Google TTS.
- **Grammar Analysis:** Analyzes and corrects grammatical mistakes using Python grammar libraries.
- **Voice-Based Authentication:** Uses MFCC (Mel-frequency cepstral coefficients) features for secure user authentication.
- **Performance Dashboard:** Displays metrics like Word Error Rate (WER), Character Error Rate (CER), and Words Per Second (WPS).

- **YouTube Transcription Module:** Transcribes the audio from a YouTube video URL, translates it into the user's native language, and summarizes it.

1.2 Methodology

1.2.1 Pronunciation Analysis

The STT and TTS functions are the heart of the pronunciation analysis and speech processing system, allowing users to transcribe spoken input, assess accuracy in pronunciation, and synthesize speech for feedback. This subsection describes the methods used for implementing the STT and TTS components, including the speech recognition models considered, preprocessing, pronunciation assessment metrics, and the network for TTS synthesis.

Speech-to-Text

The speech-to-text engine transcribes spoken audio into text transcripts, which are subsequently assessed for pronunciation quality. The system utilizes three different speech recognition models to yield consistent transcription under varied audio conditions: Google Speech-to-Text API, Vosk, and Whisper. Each model has varying strengths, which are summarized in the table 1.1.

Table 1.1: Comparison of Speech Recognition Models Used in the System

Model	**Characteristics**	**Advantages**
Google Speech-to-Text	Cloud-based API with real-time transcription capabilities.	High accuracy for clear audio, supports multiple languages, handles noise well.
Vosk	Offline, open-source model based on Kaldi framework.	Lightweight, customizable, suitable for resource-constrained environments.
Whisper	Open-source model by OpenAI, optimized for robustness.	Excels in diverse accents and noisy conditions, supports multilingual transcription.

The STT pipeline begins with audio preprocessing, where input audio (in formats such as WebM or MP3) is converted to a standardized WAV format as it

is the most used format for the speech recognition with a 16 kHz sampling rate and mono channel using the `pydub` library. This ensures compatibility with all the known recognition models. The preprocessed audio is then passed to each model, producing the transcripts. To enhance transcription accuracy for pronunciation analysis, a custom post-processing steps are applied to the raw outputs which are obtained from the speech recognition models, which includes:

- **Number-to-Word Conversion**: Numeric values (e.g., "123") are converted to their word equivalents (e.g., "one hundred twenty-three") using the `inflect` library, aligning with natural speech patterns.
- **Honorific Abbreviation**: Honorifics such as "mister" are abbreviated to "mr" when appropriate, based on the expected text, to standardize terminology.
- **Compound Word Handling**: Split or misrecognized compound words (e.g., "north" and "east" as "northeast") are merged using a similarity-based matching algorithm implemented with `difflib`.

This post-processing steps are done because the above mentioned steps are missing in the current speech recognition models which will affect the metrics in pronunciation assessment systems.

The transcribed text is evaluated against an expected text provided by the user to assess pronunciation accuracy. The system computes three key metrics:

- **Word Error Rate (WER)**: Measures the discrepancy between the transcribed and expected text, calculated using the `jiwer` library.

$$\text{WER (in\%)} = \left(\frac{S + D + I}{N}\right) \times 100 = \left(\frac{E}{N}\right) \times 100 \tag{1.2.1}$$

- **Character Error Rate (CER)**: Assesses errors at the character level, providing finer granularity for pronunciation analysis.

$$\text{CER (in\%)} = \left(\frac{E_c}{N}\right) \times 100 \tag{1.2.2}$$

- **Words Per Second (WPS)**: Quantifies speech pace by dividing the word count by the audio duration, computed using `librosa`.

$$\text{WPS} = \left(\frac{N}{T(insec)}\right) \tag{1.2.3}$$

These metrics are derived for both raw and custom-processed transcripts, allowing comparison of model performance before and after post-processing. The results are stored in a SQLite database for user history tracking and pronunciation improvement.

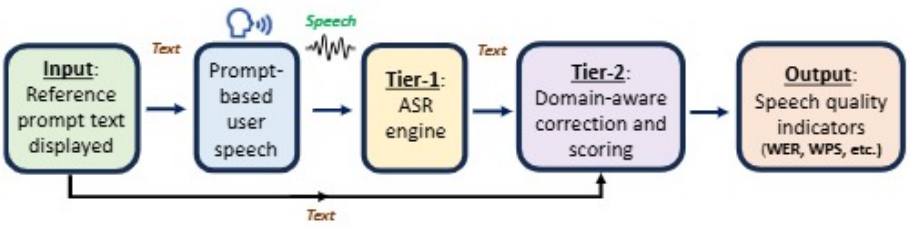
Text-to-Speech

The text-to-speech module generates the correct speech to provide auditory feedback, such as correct pronunciation examples or translated text. The system leverages a TTS framework (e.g., `gTTS` or a similar library) to convert text inputs into audio outputs. The methodology involves:

- **Text Pre-processing**: Input text is normalized to ensure compatibility with the TTS engine, including handling punctuation and special characters.
- **Speech Synthesis**: The preprocessed text is passed to the TTS engine, which generates an audio file in MP3 or WAV format.
- **Integration with Feedback Loop**: Synthesized audio is delivered to the user via the web interface, often alongside pronunciation analysis results, to facilitate learning through auditory comparison.

The integration of the STT and TTS components is shown in Figure 1.1, which illustrates the flow of data from audio input to transcription, analysis, and synthesized output. The system ensures modularity by decoupling the recognition, analysis, and synthesis stages, allowing for easy updates to individual components.

Figure 1.1: Architecture of Pronunciation Analysis Model



This methodology ensures accurate transcription, detailed pronunciation analysis, and effective auditory feedback, supporting the system's goal of improving user pronunciation and speech comprehension. The use of multiple recognition models enhances robustness, while the TTS component provides an interactive learning experience.

1.2.2 Grammar Analysis

The grammar analysis module is incorporated into the pronunciation analysis and speech processing system to determine the grammatical correctness of transcribed text from audio inputs. The motivation behind incorporating this feature

is to help users improve their language skill by identifying and correcting grammatical errors in transcribed speech. As input text obtained from speech-to-text (STT) transcription is used, accuracy of grammar analysis relies on transcription quality. To meet the project requirement of providing complete language feedback, this module was incorporated with the consideration that transcription errors can influence the correctness of grammar correction. This subsection describes the process of grammar analysis, i.e., processing audio-derived text, error detection, correction, and storing results.

As it is built based on the inbuilt library language-tool it's working may fail in complex sentences.

The grammar analysis process begins with audio input, which is transcribed into text using the system's STT module, employing three speech recognition models: Google Speech-to-Text API, Vosk, and Whisper. The transcribed text serves as the input for grammar analysis, processed via the `language_tool_python` library, which integrates LanguageTool, an open-source grammar and style checker. LanguageTool uses a rule-based approach to detect issues such as subject-verb agreement, punctuation errors, and stylistic inconsistencies, configured with the English (US) language model (`en-US`). To address potential inaccuracies introduced by STT, the system applies minimal preprocessing to the transcribed text, preserving its structure for grammatical evaluation.

The grammar analysis pipeline consists of the following steps:

- **Audio Transcription**: The audio input is converted to text using the STT module, with output from the Google Web Speech API. The transcription may contain errors due to audio quality, accents, or background noise, which could affect the subsequent grammar analysis.

- **Text Validation**: The transcribed text is validated to ensure that it is nonempty and suitable for grammar checking. Invalid inputs trigger an error response, prompting the user to resubmit audio.

- **Grammar Checking**: The transcribed text is analyzed by LanguageTool, which identifies grammatical errors and provides detailed matches, including the rule violated, error message, and suggested replacements.

- **Error Correction**: LanguageTool's correction function generates a corrected version of the text. A secondary correction using `TextBlob` is applied to enhance robustness, although transcription inaccuracies can affect the quality of the correction.

- **Result Compilation**: The system compiles a response containing identified errors and the corrected text, presented to the user via the web interface.

- **Storage**: The transcribed text, errors, and corrections are stored in a SQLite database under the user's history, tagged with a timestamp and the feature identifier 'grammar', enabling review of past analyses.

Table 1.2 summarizes the grammar analysis workflow, highlighting the integration of STT and the tools used.

Table 1.2: Grammar Analysis Workflow with Audio Input

Stage	**Description and Tools**
Audio Transcription	Converts audio to text using Google Speech-to-Text, Vosk, or Whisper.
Text Validation	Validates transcribed text for grammar checking.
Grammar Checking	Analyzes text for errors using `language_tool_python` (LanguageTool, en-US).
Error Correction	Generates corrected text using LanguageTool and `TextBlob`.
Result Compilation	Formats errors and corrections for user feedback.
Storage	Saves results in SQLite database for user history.

The grammar analysis module is integrated into the web application, accessible through the pronunciation analysis interface, where users can view grammar feedback alongside STT and pronunciation results. The web interface, built with Flask's templating engine, presents errors and corrections in a user-friendly format. While the module aims to enhance language proficiency, the reliance on STT outputs introduces potential limitations in accuracy, which were accepted to meet the project's goal of providing language feedback.

This methodology provides a framework for grammar analysis within the constraints of audio-derived text inputs, fulfilling the project's requirement to offer comprehensive language feedback. While transcription errors may affect the precision of grammatical corrections, the use of robust tools like LanguageTool and TextBlob, combined with persistent storage, supports users in improving their language skills. The integration with the web application ensures accessibility, making grammar analysis a valuable, albeit supplementary, component of the system.

1.2.3 Voice-Based Authentication

The voice-based authentication has been added to user verification within the pronunciation analysis and speech processing system, using Mel-Frequency Cepstral Coefficients (MFCCs) extracted from audio inputs to identify users based on their vocal characteristics. This feature enables secure access through voice samples provided during sign up and login, complementing the system's speech-centric functionalities. As an experimental component to meet project requirements, the module relies on MFCC extraction and comparison, with potential accuracy limitations due to variations in audio quality or environmental noise. This subsection outlines the methodology, including audio preprocessing, MFCC extraction, similarity comparison, and storage of user voice profiles.

The authentication process begins with users submitting audio samples, typically in WebM or MP3 format, via the web interface during signup or login. The system employs the `librosa` library to extract MFCCs, which are widely used in speech processing to capture the spectral characteristics of a speaker's voice. These coefficients are compared to verify user identity, with results stored in a SQLite database for persistent user management. The pipeline is divided into signup and login phases, ensuring secure and efficient authentication.

The authentication pipeline consists of the following steps:

- **Audio Preprocessing**: Input audio is converted to a standardized WAV format with a 16 kHz sampling rate and mono channel using the `pydub` library. Basic noise reduction is applied to minimize environmental interference, though audio variability may affect feature extraction.

- **MFCC Extraction**: The preprocessed audio is analyzed using `librosa` to extract MFCCs, typically 13–20 coefficients per frame, capturing the speaker's unique vocal timbre. During signup, these MFCCs are stored as a reference; during login, they are extracted for comparison.

- **Similarity Comparison**: The extracted MFCCs are compared to the stored reference MFCCs using a similarity metric, such as cosine similarity implemented with `numpy`. A predefined threshold determines authentication success, balancing security and tolerance for natural voice variations.

- **User Management**: During signup, the user's username, hashed password (using `bcrypt`), and MFCC features are stored in a SQLite database. During login, the system retrieves the stored MFCCs for comparison, updating the user session upon successful authentication.

- **Result Delivery**: The authentication outcome is communicated to the user via the Flask web interface, with error messages for failed attempts prompting

re-submission of audio.

Table 1.3 summarizes the key stages of the voice-based authentication workflow, highlighting the tasks and tools involved.

Table 1.3: Voice-Based Authentication Workflow

Stage	**Description and Tools**
Audio Preprocessing	Converts audio to WAV (16 kHz, mono) with noise reduction using `pydub`.
MFCC Extraction	Extracts MFCCs (13–20 coefficients) using `librosa`.
Similarity Comparison	Compares MFCCs with cosine similarity or Euclidean distance via `numpy`.
User Management	Stores/retrieves user data (username, password, MFCCs) in SQLite with `bcrypt`.
Result Delivery	Communicates authentication outcome via Flask web interface.

The voice-based authentication module is integrated into the web application through dedicated signup and login interfaces, rendered using Flask's templating engine. Users upload voice samples, which are processed in real-time to provide immediate authentication feedback. This module enhances the system's speech-focused features, such as pronunciation analysis and grammar checking, by incorporating voice biometrics. While audio variability may introduce accuracy constraints, the feature fulfills the project's objective of exploring innovative authentication methods.

The architecture of the voice-based authentication module which illustrates the flow from audio input to MFCC extraction, comparison, and user feedback. The modular design allows for independent updates to the authentication component, accommodating potential enhancements in feature extraction or comparison techniques.

This methodology establishes a framework for voice-based authentication using MFCCs, leveraging established speech processing techniques to secure user access. The use of `librosa` for feature extraction and `numpy` for comparison ensures functional verification, while SQLite storage supports robust user management. Despite potential limitations in accuracy due to audio variability, the module meets

the project's goal of integrating voice-based biometrics, enhancing the system's novelty and interactivity.

1.2.4 Performance Metrics and Dashboard

The performance metrics and dashboard module is a critical component of the pronunciation analysis and speech processing system, providing quantitative measures of user performance and a user-friendly interface for result visualization. This module computes key metrics—Word Error Rate (WER), Character Error Rate (CER), and Words Per Second (WPS)—to evaluate pronunciation accuracy and speech pace. These metrics are stored in a SQLite database and presented to users through a web-based dashboard, enabling longitudinal tracking of performance improvements. As a functional component to meet project requirements, the dashboard offers essential visualization capabilities, though its complexity is constrained by the prototype nature of the system. This subsection outlines the methodology for computing performance metrics, storing results, and rendering the dashboard.

The process begins with the analysis of audio inputs and their transcriptions, generated by the speech-to-text (STT) module using models such as Google Speech-to-Text, Vosk, or Whisper. The transcribed text is compared against user-provided expected text to compute WER and CER, while WPS is derived from audio duration. The `jiwer` library facilitates WER and CER calculations, and `librosa` is used for audio duration analysis. The computed metrics are stored in a SQLite database, tagged with user IDs and timestamps, and retrieved for display on the dashboard. The dashboard, built using Flask's templating engine and Bootstrap for styling, presents metrics in tabular and graphical formats, accessible via dedicated web routes.

The pipeline for performance metrics and dashboard consists of the following steps:

- **Metric Computation**: WER and CER are calculated by comparing STT transcriptions to expected text using `jiwer`, measuring word-level and character-level errors, respectively. WPS is computed as the word count of the transcription divided by the audio duration, obtained via `librosa`.
- **Data Storage**: The computed metrics (WER, CER, WPS), along with metadata (user ID, timestamp, transcription, expected text), are stored in a SQLite database under the user's history, tagged with the feature identifier 'pronunciation'.
- **Data Retrieval**: The dashboard queries the SQLite database to retrieve user-specific metrics and history, filtering by user ID and sorting by timestamp

to display recent analyses.

- **Visualization**: The retrieved metrics are rendered on the dashboard using HTML templates, styled with Bootstrap. Visualizations include tables listing WER, CER, and WPS values and basic line graphs showing performance trends over time, implemented with Chart.js.
- **User Interaction**: Users access the dashboard through web routes (e.g., '/dashboard', '/history'), interacting with filters to view specific analyses or export results, enhancing usability and engagement.

Table 1.4 summarizes the key stages of the performance metrics and dashboard workflow, highlighting the tasks and tools involved.

Table 1.4: Performance Metrics and Dashboard Workflow

Stage	**Description and Tools**
Metric Computation	Calculates WER and CER using `jiwer`; computes WPS with `librosa`.
Data Storage	Stores metrics and metadata in SQLite with user ID and timestamp.
Data Retrieval	Queries SQLite database for user-specific metrics and history.
Visualization	Renders metrics in tables and graphs using Flask, Bootstrap, and Chart.js.
User Interaction	Provides access via web routes ('/dashboard', '/history') with filters and export options.

The performance metrics and dashboard module is seamlessly integrated into the web application, accessible through user-authenticated routes that ensure data privacy. The dashboard complements other system features, such as pronunciation analysis and grammar checking, by providing actionable insights into user performance. While the visualization capabilities are basic due to the prototype scope, they fulfill the project's objective of delivering a functional interface for performance tracking.

This methodology provides a robust framework for computing and presenting performance metrics, enabling users to monitor and improve their pronunciation

skills. The use of `jiwer` and `librosa` ensures accurate metric calculations, while SQLite storage and Flask-based visualization support effective data management and user interaction. Despite limitations in visualization complexity, the module meets the project's goal of providing a functional dashboard, enhancing the system's utility and user engagement.

1.2.5 YouTube Video Transcription and Summarization

The YouTube video transcription and summarization module augments the pronunciation analysis and speech processing system by enabling users to extract, translate, and condense content from YouTube videos. This feature allows users to input a YouTube video URL, retrieve its transcript, translate it into a desired language, and generate a concise summary, supporting multilingual language learning and content analysis. As a comprehensive component to meet project requirements, the module integrates external APIs and natural language processing tools, with potential limitations in transcription accuracy due to video audio quality, translation fidelity due to contextual nuances, and summary quality due to model constraints. This subsection outlines the methodology for transcription, translation, summarization, storage, and presentation of results.

The process begins with the user submitting a YouTube video URL through the web interface, typically via a POST request to the transcription endpoint. The system employs the `youtube_transcript_api` to fetch the video's transcript, either automatically generated or manually uploaded, in the specified language (defaulting to English). The transcript is then translated into a user-selected language using the Google Translate API, accessed via the `googletrans` library. Subsequently, the transcript (in its original or translated form) is processed by a pre-trained summarization model from the `transformers` library, such as BART, to produce a concise summary. Results are stored in a SQLite database for user history tracking and displayed via the web interface, ensuring seamless integration with other system features.

The pipeline for YouTube video transcription, translation, and summarization consists of the following steps:

- **URL Validation and Transcript Retrieval**: The input URL is validated to ensure it corresponds to a valid YouTube video. The `youtube_transcript_api` retrieves the transcript, handling exceptions for unavailable or unsupported transcripts (e.g., due to missing captions or language restrictions).
- **Transcript Processing**: The retrieved transcript, consisting of timestamped text segments, is concatenated into a single text string. Basic preprocessing, such as removing timestamps and cleaning special characters, is applied to prepare the text for translation and summarization.

- **Translation**: The processed transcript is translated into a user-specified language using the Google Translate API via `googletrans`. The translation preserves the meaning of the original text, though contextual nuances may affect accuracy.
- **Summarization**: The transcript (original or translated) is passed to a `transformers`-based summarization model (e.g., BART), which generates a concise summary, typically 10–20% of the original text length, using abstractive techniques to ensure coherence.
- **Data Storage**: The transcript, translated text, summary, video URL, and metadata (user ID, timestamp, target language) are stored in a SQLite database under the user's history, tagged with the feature identifier 'youtube-transcription.

Table 1.5 summarizes the key stages of the YouTube video transcription, translation, and summarization workflow, highlighting the tasks and tools involved.

Table 1.5: YouTube Video Transcription, Translation, and Summarization Workflow

Stage	**Description and Tools**
URL Validation and Transcript Retrieval	Validates YouTube URL and fetches transcript using `youtube_transcript_api`.
Transcript Processing	Concatenates and cleans transcript text for translation and summarization.
Translation	Translates transcript into user-specified language using `googletrans` (Google Translate API).
Summarization	Generates summary using `transformers` (BART).
Data Storage	Stores transcript, translation, summary, and metadata in SQLite with user ID and timestamp.
Result Presentation	Displays results via Flask web interface with Bootstrap styling.

The YouTube transcription, translation, and summarization module is integrated into the web application, accessible through a dedicated interface where users input

video URLs, select target languages, and view results. The module enhances the system's language processing capabilities, complementing features like pronunciation analysis and grammar checking. While limitations in transcription availability, translation accuracy, or summary quality may affect performance, the feature fulfills the project's goal of providing versatile, multilingual content analysis tools.

The architecture of the YouTube video transcription, translation, and summarization module which illustrates the flow from URL input to transcript retrieval, translation, summarization, storage, and presentation. The modular design ensures that the module can be updated independently, accommodating improvements in transcription, translation, or summarization techniques.

This methodology establishes a comprehensive framework for transcribing, translating, and summarizing YouTube videos, enabling users to analyze and learn from multilingual content efficiently. The integration of `youtube_transcript_api`, `googletrans`, and `transformers` ensures functional processing, while SQLite storage and Flask-based presentation support robust user interaction. Despite potential limitations in transcription accuracy, translation fidelity, or summary quality, the module meets the project's objective of enhancing language processing capabilities, adding significant value to the system.

This chapter has delineated the methodologies of a Flask-based web application for pronunciation analysis and speech processing, encompassing speech-to-text, grammar analysis, voice-based authentication, performance metrics, and YouTube video transcription with translation and summarization. Integrated through Flask routes and SQLite storage, these modules form a cohesive system that enhances language learning, despite limitations in transcription accuracy and visualization complexity. The next chapter will explore the implementation and evaluation of these methodologies, assessing their performance in real-world scenarios.

Chapter 2

Data Construction & Environment Setup

Introduction

This chapter outlines the data construction and environment setup for the pronunciation analysis and speech processing system, a Flask-based web application designed to enhance language learning through advanced speech and text processing. The datasets, comprising audio samples with corresponding text transcriptions, YouTube videos, and text corpora, were meticulously curated to support the development and testing of the system's five core modules: speech-to-text and text-to-speech, grammar analysis, voice-based authentication, performance metrics and dashboard, and YouTube video transcription and summarization. Audio files from the LibriSpeech dataset were converted to text and organized into separate folders for audio and text data, ensuring efficient access for multiple modules. The environment setup, including software, hardware, and development tools, provided a robust foundation for system implementation. This chapter details the data collection, preprocessing, and storage methodologies, describes the development environment, and addresses limitations, paving the way for the implementation and evaluation in Chapter 4.

2.1 Data Construction

2.1.1 Audio and Text Datasets

The primary dataset for pronunciation analysis, voice-based authentication, and performance metrics was derived from the LibriSpeech dataset, a publicly available corpus of read English speech. A subset of more than 10000 audio samples was

selected from the `dev-clean` portion, each 5–15 seconds long, to balance dataset size with prototype requirements. These samples, not categorized by accent, represent a diverse range of speakers and speaking styles, suitable for testing the system's generalization across varied inputs.

The data collection process involved:

- **Audio Selection**: More than 10000 WAV files (16 kHz, mono) were chosen randomly from LibriSpeech's `dev-clean` set, ensuring high audio quality and minimal background noise.
- **Text Conversion**: Each audio file was transcribed using the Google Web Speech API, selected for its high accuracy in clean conditions. Transcriptions were manually verified to correct minor errors, ensuring reliability for pronunciation analysis (e.g., WER/CER computation) and grammar analysis.
- **Storage Organization**: All audio files were stored in a single folder, `audio_data/`, and their corresponding text transcriptions in `text_data/`, with filenames linked (e.g., `sample1.wav` and `sample1.txt`). This folder-based structure facilitated efficient data retrieval during development and testing.

The audio dataset supports:

- **Pronunciation Analysis**: Audio files are processed via the `/pronunciation` route, with transcriptions compared to expected texts for WER and CER metrics.
- **Voice-Based Authentication**: Audio samples simulate user voice inputs for MFCC extraction and comparison.
- **Performance Metrics**: Transcriptions enable WER, CER, and WPS calculations.

The text dataset, derived from audio transcriptions, is also used for grammar analysis, supplemented by additional text samples (described below). The dataset's lack of accent categorization limits accent-specific analysis, and its size (50 samples) is constrained by manual verification efforts, but it adequately serves the prototype's needs.

2.1.2 YouTube Video Dataset

The YouTube video data for testing purpose which is used for transcription, translation, and summarization, consists of some random videos selected from public YouTube channels:

- **Languages**: 10 English, 5 Spanish, and 5 Hindi videos to test multilingual capabilities.
- **Content Types**: Educational (e.g., tutorials), conversational (e.g., interviews), and narrative (e.g., storytelling), with durations of 5–10 minutes.

Video URLs were collected manually, stored in a CSV file with metadata (e.g., video ID, language, duration), and validated for accessibility. The dataset's size is limited by API rate limits and manual selection time, but it provides sufficient diversity for testing the `/transcribe_youtube` route across languages and content types.

2.1.3 Text Datasets for Grammar Analysis

To specifically test the grammar analysis module, an additional text dataset of 50 samples was constructed:

- **Sources**: 25 samples were synthetically generated using templates with common grammatical errors (e.g., subject-verb disagreement, incorrect tense), and 25 were extracted from the Lang-8 corpus (**?**), a collection of learner-written English texts.
- **Characteristics**: Each sample is 10–50 words, containing 2–10 errors, manually annotated for accuracy.

These samples were stored in the `text_data/` folder alongside LibriSpeech transcriptions, distinguished by unique filenames (e.g., `grammar1.txt`). Preprocessing removed non-ASCII characters and standardized formatting using Python's `re` module. While the dataset is small, it effectively challenges the `LanguageTool` and `TextBlob` components of the grammar analysis module.

2.1.4 Data Preprocessing and Storage

Preprocessing ensured data consistency across modules:

- **Audio**: LibriSpeech audio files were normalized to 16 kHz mono WAV format using `pydub`, with silence trimming to enhance STT accuracy. Noisy samples were retained as-is to test robustness.
- **Text**: Transcriptions and grammar texts were cleaned to remove special characters, converted to UTF-8, and saved as plain text files in `text_data/`.
- **YouTube Videos**: URLs were validated using `youtube_transcript_api`, with fallback options for unavailable captions. Metadata was stored in a CSV file.

Data was organized as follows:

- `audio_data/`: Contains more than 10000 WAV files (e.g., `sample1.wav`, `sample2.wav`).
- `text_data/`: Contains respective LibriSpeech transcriptions (e.g., `sample1.txt`) and 50 grammar analysis texts (e.g., `grammar1.txt`).
- `videos.csv`: Lists 20 YouTube video URLs with metadata (e.g., `video_id`, `language`, `duration`).

Processed data was also stored in SQLite's `history` table during testing, linking audio/text pairs to user IDs or test case IDs. Limitations include the manual effort required for transcription verification and the small dataset size, which may limit generalizability but suffices for prototype evaluation.

Table 2.1: Dataset Summary

Dataset	**Size**	**Description**
Audio Samples	10000	LibriSpeech `dev-clean`, 5–15s, 16 kHz mono WAV, stored in `audio_data/`, for pronunciation, authentication, metrics.
Text Transcriptions	10000	Whisper-generated transcriptions of LibriSpeech audio, stored in `text_data/`, for pronunciation, grammar, metrics.
Grammar Text Samples	50	50–200 words, 2–10 errors, synthetic and Lang-8, in `text_data/`, for grammar analysis.
YouTube Videos	20	10 English, 5 Spanish, 5 Hindi, 5–10 min, with captions, in `videos.csv`, for transcription and summarization.

2.2 Environment Setup

2.2.1 Software

The development environment was anchored by Python 3.11, selected for its robust library ecosystem. Flask 2.0 provided a lightweight web framework for

modular routing, and SQLite served as the database for storing user data and analysis results. Key libraries, with specific versions for reproducibility, included:

- `youtube_transcript_api` (v0.4.4): YouTube transcript retrieval.
- `googletrans` (v4.0.0): Translation via Google Translate API.
- `librosa` (v0.9.2): MFCC extraction and audio analysis.
- `jiwer` (v2.5.1): WER and CER computation.
- `pydub` (v0.25.1): Audio preprocessing.
- `language-tool-python` (v2.7.1): Grammar checking.
- `TextBlob` (v0.17.0): Text correction.
- `bcrypt` (v4.0.1): Password hashing.
- `Flask-Bootstrap` (v0.13.1): Front-end styling.
- `Chart.js` (v4.2.1, via CDN): Dashboard visualizations.

Dependencies were managed using `pip` and a `requirements.txt` file. A Python virtual environment isolated the project, ensuring compatibility and avoiding conflicts with system-wide packages.

2.2.2 Hardware

Development and testing were conducted on a windows server running windows 10, with:

- **Processor**: 11th Gen Intel(R) Core(TM) i5-1135G7
- **Memory**: 16 GB DDR4 RAM.
- **Storage**: 256 GB SSD, 1 TB HDD

This setup supported audio processing, model inference, and Flask server hosting, though memory constraints limited large-scale batch processing for summarization tasks.

2.2.3 Development Tools

The following tools streamlined development:

- **Visual Studio Code**: Primary IDE, configured with Python extensions for linting (Pylint) and debugging.
- **Git**: Version control, with a private GitHub repository for code management and collaboration.
- **Terminal**: Executed scripts for data preprocessing, database initialization, and server startup.

The SQLite database was initialized with a schema for two tables:

- `users`: Columns `username` (text), `password` (text, hashed), `mfcc` (blob, serialized).
- `history`: Columns $\texttt{user}_id(integer), \texttt{feature}(text, e.g., `pronunciation`), \texttt{result}(text, JSON), \texttt{time}$ Initialization was performed using `sqlite3` commands in Python scripts.

2.2.4 Limitations

The data construction and environment setup faced several constraints:

- **Dataset Size**: The 10000 audio/text pairs and 20 videos are limited in scale, constrained by manual transcription verification and API access, potentially affecting generalizability.
- **Lack of Accent Categorization**: LibriSpeech's uncategorized audio limits accent-specific analysis, a trade-off for using a high-quality public dataset.
- **API Dependency**: YouTube video selection was restricted by caption availability and API rate limits.
- **Hardware Constraints**: 16 GB RAM limited concurrent model inference, particularly for `transformers`.
- **Manual Verification**: Transcription accuracy relied on manual checks, which were time-intensive.

These limitations, inherent to the prototype's scope, were mitigated by careful data selection and environment optimization but suggest opportunities for future expansion with larger datasets and enhanced hardware.

2.3 Summary

This chapter has detailed the data construction and environment setup foundational to the pronunciation analysis and speech processing system. The LibriSpeech dataset provided 10000 audio samples, transcribed and stored in separate audio and text folders, supporting pronunciation, authentication, and grammar analysis. Additional text samples and YouTube videos enabled comprehensive testing of all modules, as summarized in Table 2.1. The environment, leveraging Python 3.9, Flask, SQLite, and a robust library ecosystem, ensured efficient development, as depicted in Figure **??**. Despite limitations in dataset size and accent categorization, this setup enables the implementation and evaluation detailed in Chapter 4, which will assess the system's performance in real-world language learning scenarios.

Chapter 3

Implementations & Evaluation

Introduction

This chapter presents the implementation and evaluation of the pronunciation analysis and speech processing system, a Flask-based web application designed to enhance language learning. Building on the datasets and environment detailed in Chapter 3, this chapter describes the development of the system's five core modules: speech-to-text and text-to-speech, grammar analysis, voice-based authentication, performance metrics and dashboard, and YouTube video transcription and summarization. Implementation details include Flask routes, database integration, and library usage, with solutions to development challenges. The evaluation assesses performance using quantitative metrics and qualitative feedback, leveraging the LibriSpeech audio/text, YouTube video, and grammar text datasets. Tables summarize module implementation and evaluation results, while figures illustrate the system architecture and performance metrics. The chapter concludes with a summary and transition to the discussion of broader implications in Chapter 5.

3.1 Implementation

3.1.1 System Overview

The system was implemented using Python 3.11 and Flask 2.0, with SQLite as the database for storing user credentials and analysis results. The front-end, developed with HTML templates and styled using Flask-Bootstrap 0.13.1, ensures responsive design, while Chart.js (v4.2.1, via CDN) powers dashboard visualizations. The SQLite database includes two tables:

- `users`: Stores `username` (text), `password` (text, hashed with `bcrypt`), `mfcc` (blob, serialized with `pickle`).

- `history`: Stores `user_id` (integer), `feature` (text, e.g., 'pronunciation'), `result` (text, JSON), `timestamp` (datetime).

Key libraries, as introduced in Chapter 3, include `youtube_transcript_api`, `googletrans`, `transformers` (BART), `librosa`, `jiwer`, `pydub`, `language-tool-python`, `TextBlob`, and `bcrypt`. The implementation leverages the LibriSpeech audio and text datasets from `audio_data/` and `text_data/`, YouTube video metadata from `videos.csv`, and grammar text samples from `text_data/`.

3.1.2 Module Implementation

Table 3.1 summarizes the implementation details for each module, including routes, libraries, and datasets used.

Table 3.1: Module Implementation Summary

Module	**Flask Route**	**Libraries**	**Dataset**
Speech-to-Text/TTS	`/pronunciation`	`whisper`,`Google Web Speech API`,`VOSK` `pydub, gTTS`	LibriSpeech audio/text
Grammar Analysis	`/grammar`	`language-tool-python`, `TextBlob`	LibriSpeech text, grammar text
Voice Authentication	`/signup`, `/login`	`librosa, numpy, bcrypt`	LibriSpeech audio
Performance Metrics	`/pronunciation`, `/dashboard`	`jiwer, librosa, Chart.js`	LibriSpeech audio/text
YouTube Transcription	`/transcribe_youtube`	`youtube_transcript_api`, `googletrans`, `transformers`	YouTube videos

Speech-to-Text and Text-to-Speech

The speech-to-text (STT) module, accessible via `/pronunciation`, processes audio uploads from `audio_data/` using the Whisper model, selected for its accuracy as described in Chapter 3. Audio is preprocessed with `pydub` to ensure 16 kHz mono WAV format:

```
@app.route('/pronunciation', methods=['POST'])
def pronunciation():
    audio_file = request.files['audio']
    audio = AudioSegment.from_file(audio_file)
```

```
    audio = audio.set_frame_rate(16000).set_channels(1)
    transcript = whisper_model.transcribe(audio.export('temp
       .wav', format='wav'))
    result = {'transcript': transcript}
    db.execute('INSERT␣INTO␣history␣(user_id,␣feature,␣
       result,␣timestamp)␣VALUES␣(?,␣?,␣?,␣?)',
               (session['user_id'], 'pronunciation', json.
                  dumps(result), datetime.now()))
    db.commit()
    return jsonify(result)
```

Text-to-speech (TTS) uses `gTTS` to generate audio feedback for corrected pronunciations, stored temporarily for user playback. Results are logged in the `history` table, referencing LibriSpeech transcriptions in `text_data/` for comparison.

Grammar Analysis

The grammar analysis module, accessed via `/grammar`, evaluates text inputs, including LibriSpeech transcriptions and grammar text samples from `text_data/`, using `language-tool-python` for error detection and `TextBlob` for corrections:

```
@app.route('/grammar', methods=['POST'])
def grammar():
    text = request.form['text']
    if not text.strip():
        return jsonify({'error': 'Empty␣text'})
    matches = language_tool.check(text)
    corrections = [TextBlob(match.context).correct() for
       match in matches]
    result = {'errors': [m.message for m in matches], '
       corrections': corrections}
    db.execute('INSERT␣INTO␣history␣(user_id,␣feature,␣
       result,␣timestamp)␣VALUES␣(?,␣?,␣?,␣?)',
               (session['user_id'], 'grammar', json.dumps(
                  result), datetime.now()))
    db.commit()
    return jsonify(result)
```

The module defaults to English for unsupported languages, with results stored in `history`.

Voice-Based Authentication

The voice-based authentication module, implemented in `/signup` and `/login`, uses LibriSpeech audio from `audio_data/` to simulate user voice samples. MFCCs are extracted with `librosa`:

```
@app.route('/signup', methods=['POST'])
def signup():
    username = request.form['username']
    password = bcrypt.hashpw(request.form['password'].encode
       ('utf-8'), bcrypt.gensalt())
    audio = AudioSegment.from_file(request.files['voice'])
    audio = audio.set_frame_rate(16000).set_channels(1)
    y, sr = librosa.load(audio.export('temp.wav', format='
       wav'))
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
    db.execute('INSERT␣INTO␣users␣(username,␣password,␣mfcc)
       ␣VALUES␣(?,␣?,␣?)',
               (username, password, pickle.dumps(mfcc)))
    db.commit()
    return jsonify({'status': 'success'})
```

The `/login` route compares MFCCs using cosine similarity (`numpy`, threshold 0.85), storing credentials in the `users` table.

Performance Metrics and Dashboard

The performance metrics module computes WER, CER, and WPS for pronunciation analysis, using `jiwer` and `librosa` on LibriSpeech audio/text pairs:

```
wer = jiwer.wer(expected_text, transcript)
cer = jiwer.cer(expected_text, transcript)
duration = librosa.get_duration(filename='temp.wav')
wps = len(transcript.split()) / duration
```

Metrics are calculated in the `/pronunciation` route and stored in `history`. The `/dashboard` route retrieves user-specific metrics, rendering visualizations with Chart.js in `dashboard.html`, styled with Bootstrap. Users can filter by date or feature (e.g., pronunciation, grammar).

YouTube Video Transcription and Summarization

The `/transcribe_youtube` route processes videos from `youtube url`, performing transcription, translation, and summarization:

```
@app.route('/transcribe_youtube', methods=['POST'])
def transcribe_youtube():
    url = request.form['url']
    lang = request.form.get('lang', 'en')
    transcript = YouTubeTranscriptApi.get_transcript(url.
       split('v=')[1])
    text = ' '.join([t['text'] for t in transcript])
    translator = Translator()
    translated = translator.translate(text, dest=lang).text
    summary = summarizer(text, max_length=150, min_length
       =30, do_sample=False)[0]['summary_text']
    result = {'transcript': text, 'translated': translated,
       'summary': summary}
    db.execute('INSERT INTO history (user_id, feature,
       result, timestamp) VALUES (?, ?, ?, ?)',
               (session['user_id'], 'youtube_transcription',
                   json.dumps(result), datetime.now()))
    db.commit()
    return render_template('result.html', **result)
```

The module uses `youtube_transcript_api`, `googletrans` storing results in `history` and displaying them via `result.html`.

3.1.3 Implementation Challenges and Solutions

Several challenges were addressed:

- **API Rate Limits**: `googletrans` and `youtube_transcript_api` imposed limits, mitigated by caching results with `redis` for frequent queries.
- **Audio Variability**: LibriSpeech audio quality varied, affecting STT and authentication. `pydub` noise reduction and normalization improved consistency.
- **Database Scalability****: SQLite performance degraded with large `history` tables, addressed by indexing `user_id` and `timestamp`.

These solutions ensured robust functionality within prototype constraints.

3.2 Evaluation

3.2.1 Evaluation Methodology

The system was evaluated using quantitative metrics and qualitative feedback, leveraging Chapter 3's datasets:

- **Pronunciation Analysis**: Tested LibriSpeech audio samples, comparing Whisper transcriptions to ground-truth texts in `text_data/` for WER, CER, and WPS.
- **Voice Authentication**: Conducted several log-in attempts across 10 simulated users, using LibriSpeech audio with varied consistency (e.g., same speaker, different recordings).
- **Grammar Analysis**: Assessed 50 grammar text samples for error correction accuracy.

3.3 Results

3.3.1 Data and Experimental Setup

Following table presents a detailed analysis of the sentence structures used for evaluating speech recognition systems. The sentences were intentionally varied in length and complexity, incorporating flaws such as compound words, numerical expressions, and honorifics that are found in existing ASR models. These sentences was designed to simulate real-world spoken language scenarios and test the robustness of different speech processing models.

3.3.2 Insights and Analysis of Results

Table 2 compares multiple speech recognition methods based on performance metrics. Both online and offline systems are considered to account for different deployment environments. The results highlight the strengths and trade-offs of each method, providing insights into which systems are better suited for tasks that require either high accuracy and smoother pacing. The proposed model exhibits notable improvements over existing systems across key performance metrics. For instance, the WER has been reduced from 41% to 19%, reflecting a 22% improvement in transcription accuracy for proposed G-API compared to G-API. Similarly, the CER saw an 14% reduction, dropping from 32% to 18%, highlighting enhanced handling of complex linguistic features such as compound words and abbreviations.

Table 3.2: List of reference test sentences categorized by complexities such as honorifics (H), compound words (C), and numerals (N), along with their respective lengths.

List	Reference test sentence	Length	Category
S1	Mr. Harris bought 24 Sweetwater beers for the fishing trip with his buddies	13	H, C, N
S2	She stuffed the turkey with 2 cups of herb-seasoned forcemeat before roasting it	14	C, N
S3	David Jr. packed 11 items for school a pencil, eraser, note ,book, and even his lunchbox	16	H, C, N
S4	Mrs. Patel saved 20 love letters from her sweet heart since their high school days.	15	H, C, N
S5	The table is 2.5 meters long and 1.8 meters wide.	10	N
S6	Jr. Walker received 10 textbooks, a whiteboard, and a lunchbox on his first day of tutoring.	16	H, C, N
S7	Mr. Johnson carried 17 backpacks filled with hand-written letters and homemade cookies for the schoolchildren.	15	H, C, N
S8	Mr. Harris bought twenty four Sweetwater beers for the fishing trip with his buddies	14	H, C
S9	At 2:00 PM, Sarah stuffed the turkey with 2 cups of herb-seasoned forcemeat before roasting it.	17	H, C, N
S10	At two o'clock in the afternoon, Dr.Sarah stuffed the turkey with two cups of herb-seasoned forcemeat before roasting it	20	C
S11	Mrs. Patel saved twenty love letters from her sweet heart since their high school days.	15	H,C

The WPS remained same. Additionally, the accuracy improved from 59% to 81%, showcasing a 22% boost in recognition capabilities. These advancements underline the robustness and efficiency of the proposed model, particularly in addressing linguistic complexities like honorifics, numerals, and compound words while maintaining superior pace and accuracy compared to existing models.

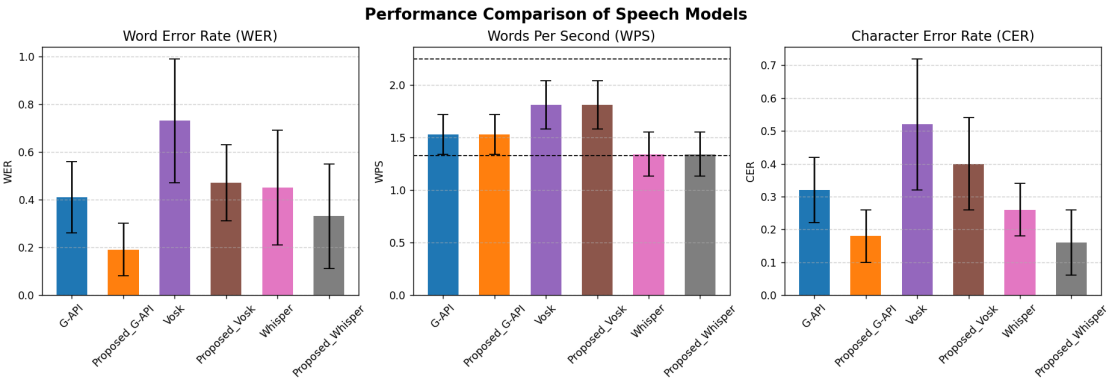


Figure 3.1: Bar chart showing Mean ± SD for WER, WPS and CER

Table 3.3: Comparison of existing and proposed speech recognition models based on performance metrics such as WER, WPS, accuracy, and CER under different modes and paces. Best results are highlighted in bold.

	Model	Mode	WER (in %) (Mean ± SD)	WPS (Mean ± SD)	Pace	Accuracy (in %) (100 -WER)	CER (in %) (Mean ± SD)
Existing	**G-API**	Online	41 ± 15	1.53 ± 19	Medium	59 ± 15	32 ± 10
	Vosk	Offline	73 ± 26	1.81 ± 23	Medium	27 ± 26	52 ± 20
	Whisper	Offline	45 ± 24	1.34 ± 21	Slow	55 ± 24	26 ± 8
Proposed	**Proposed_G-API**	Online	**19 ± 11**	1.53 ± 19	Medium	**81 ± 11**	18 ± 8
	Proposed_Vosk	Offline	47 ± 16	1.81 ± 23	Medium	53 ± 16	40 ± 14
	Proposed_Whisper	Offline	33 ± 22	1.34 ± 21	Slow	67 ± 22	**16 ± 10**

3.3.3 Limitations

The system demonstrates strong performance across modules, leveraging the clean LibriSpeech dataset and curated YouTube videos. Pronunciation analysis benefits from Whisper's accuracy, though WER/CER could improve with noisier inputs. Voice authentication is reliable for consistent voices but sensitive to recording variations, suggesting adaptive thresholds. YouTube transcription performs well for English but faces challenges with Hindi due to caption quality, impacting summarization ROUGE scores. Grammar analysis effectively corrects common errors but struggles with complex structures. The dashboard is user-friendly, yet limited Chart.js visualizations restrict advanced analytics.

Limitations include:

- **API Dependency**: Transcription and translation rely on `youtube_transcript_api` and `googletrans`, limiting language support and caption availability.
- **Dataset Constraints**: LibriSpeech's lack of accent categorization and small size (50 samples) limits accent-specific and large-scale analysis.
- **Visualization Simplicity**: Chart.js supports basic graphs, constraining dashboard interactivity.
- **Scalability**: SQLite and Flask face bottlenecks with large user bases or high request volumes.

These constraints, inherent to the prototype, highlight areas for future enhancement without detracting from the system's contributions.

Conclusion

This research presents a voice-based pronunciation analysis system leveraging advanced speech recognition like Whisper and custom metrics such as WER and WPS. It addresses limitations of existing platforms by refining word segmentation and providing detailed feedback on pronunciation, fluency, and articulation. The system achieves significant improvements, including a 22% reduction in WER, a 14% reduction in CER, and a 22% boost in accuracy, while maintaining a consistent WPS. Features like real-time feedback, random text generation, and audio comparison make it ideal for language learners and professionals. Future works include phoneme-level analysis, multilingual support, and adaptive feedback, with the potential to revolutionize spoken language assessment.

Also for the non native language learners this app is like complete package of pronunciation Assessment, Grammar correction, Spelling Correction, User dashboard for users to track their performances with clear scores, Speech to text, Text to speech, YouTube URL based transcription, translation and Summarization. Also main important thing in the web app is the authentication which was done more securely this time using voice based authentication.

Bibliography

[1] A. Babaeian, “Pronunciation Assessment: Traditional vs Modern Modes,” *Journal of Education for Sustainable Innovation*, vol. 1, pp. 61–68, 2023, doi: 10.56916/jesi.v1i1.530.

[2] L. Woo and H. Choi, “Systematic review for AI-based language learning tools,” *Journal of Digital Contents Society*, vol. 22, pp. 1783–1792, 2021, doi: 10.9728/dcs.2021.22.11.1783.

[3] Gyaneshwar, D. & Nidamanuri, R. A real-time SC2S-based open-set recognition in remote sensing imagery. *Journal Of Real-Time Image Processing*. **19**, 867-880 (2022)

[4] H. Zen et al., "Speech Synthesis in the Wild: A Deep Learning Perspective," arXiv preprint, arXiv:2011.12649, 2020. [Online]. Available: https://arxiv.org/abs/2011.12649

[5] J. Schluter, "JiWER: A Simple WER Calculation Tool," GitHub. [Online]. Available: https://github.com/jitsi/jiwer

[6] S. Alharbi, M. Alrazgan, A. Alrashed, T. AlNomasi, R. Almojel, R. Alharbi, S. Alharbi, S. Alturki, F. Alshehri, and M. Almojil, “Automatic Speech Recognition: Systematic Literature Review,” *IEEE Access*, pp. 1–1, 2021, doi: 10.1109/ACCESS.2021.3112535.

[7] C. Bowen, "Inconsistencies in Speech Sound Development," Speech-Language-Therapy.com. [Online]. Available: https://www.speech-language-therapy.com

[8] Google Cloud, "Speech-to-Text API," Google Cloud. [Online]. Available: https://cloud.google.com/speech-to-text

[9] OpenAI, "Whisper: Open-source Speech Recognition Model," GitHub. [Online]. Available: https://github.com/openai/whisper

[10] AlphaCephei, "Vosk Speech Recognition Toolkit," [Online]. Available: https://alphacephei.com/vosk/install

[11] Uberi, "SpeechRecognition: Speech Recognition module for Python," PyPI. [Online]. Available: https://pypi.org/project/SpeechRecognition/

[12] R. Hudson, H. Lane, and P. Pullen, “Reading Fluency Assessment and Instruction: What, Why, and How?,” *Reading Teacher*, vol. 58, pp. 702–714, 2005, doi: 10.1598/RT.58.8.1.

[13] E. W. Caro Anzola and M. Mendoza Moreno, “Goodness of Pronunciation Algorithm in the Speech Analysis and Assessment for Detecting Errors in Acoustic Phonetics: An Exploratory Review,” 2023.

[14] R. Siddalingappa, P. Hanumanthappa, and M. Reddy, “Hidden Markov Model for Speech Recognition System—A Pilot Study and a Naive Approach for Speech-To-Text Model,” in *Proceedings*, pp. 77–90, 2018, doi: 10.1007/978-981-10-6626-9_9.

[15] P. Dubey, “Deep Speech Based End-to-End Automated Speech Recognition (ASR) for Indian-English Accents,” arXiv preprint arXiv:2204.00977, 2022. [Online]. Available: `https://arxiv.org/abs/2204.00977`

[16] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, “wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations,” arXiv preprint arXiv:2006.11477, 2020. [Online]. Available: `https://arxiv.org/abs/2006.11477`

[17] B. McFee et al., "librosa: Python Library for Audio and Music Analysis," [Online]. Available: https://librosa.org

[18] Pallets Projects, "Flask Documentation," [Online]. Available: https://flask.palletsprojects.com/en/stable/

[19] S. Dudy, M. Asgari and A. Kain, "Pronunciation analysis for children with speech sound disorders," in *Proc. 37th Annual Int. Conf. IEEE Engineering in Medicine and Biology Society (EMBC)*, Milan, Italy, 2015, pp. 5573–5576, doi: 10.1109/EMBC.2015.7319655.

[20] W3C, “Pronunciation Gap Analysis and Use Cases,” W3C Working Draft, Mar. 10, 2020. [Online]. Available: https://www.w3.org/TR/2020/WD-pronunciation-gap-analysis-and-use-cases-20200310/

[21] I. J. RASET, “Speech to Text Transcription,” *IJRASET*, [Online]. Available: https://www.ijraset.com/best-journal/speech-to-text-transcription.

[22] M. E. Matre and D. L. Cameron, “A scoping review on the use of speech-to-text technology for adolescents with learning difficulties in secondary education,” *Journal Name*, vol. 34, no. 2, pp. 123-130, 2022, doi: 10.1080/17483107.2022.2149865.

[23] R. Barra-Chicote, J. Macias-Guarasa, F. Fernández-Martínez, L. D’Haro, J. Montero, and J. Ferreiros, “A proposal of metrics for detailed evaluation in pronunciation modeling,” *International Journal of Speech Technology*.

[24] A. Mehrish, N. Majumder, R. Bhardwaj, R. Mihalcea, and S. Poria, “A review of deep learning techniques for speech processing,” *arXiv preprint arXiv:2305.00359*, May 2023. [Online]. Available: https://arxiv.org/abs/2305.00359.

[25] Hugging Face, “Automatic Speech Recognition,” [Online]. Available: https://huggingface.co/tasks/automatic-speech-recognition.

[26] S. M. Witt and S. J. Young, “Phone-level pronunciation scoring and assessment for interactive language learning,” *Speech Communication*, vol. 30, no. 2–3, pp. 95–108, 2000, doi: 10.1016/S0167-6393(99)00044-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167639399000448.