Archives available at journals.mriindia.com

International Journal on Advanced Computer Theory and Engineering

ISSN: 2319-2526

Volume 15 Issue 01, 2026

AI Automatic Pronunciation Mistake Detector

[1]Pratham Singh, [2]Rishikesh Singh, [3]Siddhi Sankhe, [4]Narendra Prajapati, [5]Manasi Churi
[1-5] Dept Of Computer Engineering, Shree L.R Tiwari College Of Engineering

Peer Review Information

Submission: 24 March 2026

Revision: 13 April 2026

Acceptance: 27 April 2026

Keywords

Automated Pronunciation Evaluation, Phoneme-Level Assessment, Multilingual ASR, Grapheme-to-Phoneme Conversion, Dynamic Time Warping

Abstract

Automated Pronunciation Evaluation plays a major role in computer-assisted learning for various languages, majorly used for learning English, and many other languages. However, effective multilingual systems for pronunciation assessment are not yet fully developed, particularly for Indic languages which have complex character and phonetic systems. Most pronunciation assessment systems utilize word-level scoring or limited acoustic models, which restrict the scope for phoneme-level assessment and accommodating linguistic diversity. Additionally, errors resulting from ASR systems affect the overall accuracy of the scoring process. This paper proposes a framework for phoneme-level pronunciation assessment system for English, Hindi, and Marathi languages. The system is developed by integrating the Whisper ASR model, word-level timestamp extraction, grapheme-to-phoneme conversion, Dynamic Time Warping for robust word alignment, and phoneme-level Levenshtein distance scoring. In addition, the schwa deletion module is included to handle Devanagari languages. The schwa deletion module is designed to eliminate the impact of schwa characters on pronunciation scores.

The framework is based on a modular three-tier structure that includes a browser-based audio capture interface, a Flask-based REST API backend, and an extensible AI processing core developed based on interface-driven model abstractions. Normalization and resampling of audio signals are performed before ASR inference to improve consistency across recording conditions, while DTW-based word alignment over a word distance matrix enhances robustness against ASR variability. The experimental results show stable word alignment against recognition noise and consistent accuracy discrimination for phoneme-level qualities across varying word pronunciation qualities. Word-level categorization and IPA visualization are provided for actionable feedback on pronunciation qualities for multilingual learning scenarios.

Introduction

Automated pronunciation assessment is one of the key tools used for computer-assisted language learning. It has been found to be a very effective tool for pronunciation assessment. Though the pronunciation assessment tools for the English language are extremely effective, the tools for the other languages, especially the Indic languages, are extremely limited. The traditional pronunciation assessment tools were based on the use of ASR confidence scores and pronunciation verification for speech recognition systems. Though the use of deep learning techniques such as the use of a combination of CTC and attention models and self-supervised representation of speech has shown promising results, the use of pronunciation assessment tools has also been

explored for specific uses. Though the word pronunciation assessment tools were extremely effective, the tools for the pronunciation assessment of phonemes were extremely limited. Recent advancements in deep learning have significantly improved pronunciation error detection. Hybrid CTC/Attention-based architectures and end-to-end neural models have demonstrated improved robustness in detecting pronunciation deviations. Similarly, self-supervised speech representation learning approaches have enhanced generalization across speakers and acoustic conditions. Despite these advancements, many of these models are language-specific, computationally intensive, and dependent on large annotated pronunciation datasets. Their scalability to low-resource or linguistically complex languages remains limited. This paper outlines a pronunciation assessment framework for phonemes for English, Hindi, and Marathi languages. This framework combines a transformer-based ASR model like Whisper with a rule-based grapheme to phoneme conversion method. Dynamic Time Warping and Levenshtein scoring are also included in this framework. In addition, schwa deletion is included for Devanagari script-based language

Background

Pronunciation is a very important component of language learning; it plays a vital role in the effective intelligibility and communication of the language. It is difficult for a non-native speaker to acquire the pronunciation of the language correctly because of phonetic variations and accents. Recently, the improvement in Automatic Speech Recognition (ASR) and Artificial Intelligence (AI) has allowed the creation of tools for pronunciation evaluation. Initially, pronunciation verification was carried out using confidence scores and acoustic likelihood scores.Recent studies have utilized deep learning techniques to improve the accuracy of pronunciation error detection. Hybrid CTC/Attention models have been found to provide better alignment and error detection accuracy. Moreover, self-supervised learning of speech representation has been found to provide better robustness to speaker and recording variability. Real-time systems for pronunciation mistake detection have been proposed to ensure better usability and accessibility. However, existing systems are mainly word-based and do not provide sufficient information at the phoneme level. Errors in ASR can also affect the accuracy of scoring methods if the alignment mechanism is not robust.

With respect to the deployment of the system, modular design and API integration are critical for the development of highly scalable AI-based systems. Despite the advances made in the field of NLP and related areas, most of the existing systems are monolingual and focused on English and other resource-rich languages. There has been a lack of emphasis on Indic languages such as Hindi and Marathi. These languages are known to have complex grapheme-phoneme correspondences and schwa deletion.

Objectives

The primary objective of this paper is to design and implement a multilingual, phoneme-level pronunciation system that provides accurate, robust and linguistically informed feedback for English, Hindi, and Marathi.To achieve this goal, the system is developed around the following core objectives:

1. To develop a modular structure for a three-tier pronunciation assessment system: For this, the system will be designed such that it includes a full-stack structure, a browser-based audio capture interface, a Flask-based REST API backend, and a modular AI core. This structure has been chosen for the project because of its flexibility and modularity. At the same time, the interface-based structure will allow the integration of different ASR or phoneme conversion tools without altering the structure of the entire system.

2. To develop a robust speech transcription: For this, the framework includes a Whisper ASR engine, which supports the functionality of speech transcription with timestamps for English, Hindi, and Marathi languages. In order to avoid the issues of transcription and ASR noise, the use of Dynamic Time Warping (DTW) has been made for the optimal word alignment between the recognized and the reference words.

3. To conduct phoneme-level pronunciation evaluation: Instead of word-level matching, the system uses grapheme-to-phoneme converters to obtain phoneme sequences of recognized and reference words in IPA format. For Devanagari script-based languages, a schwa deletion mechanism is also employed to eliminate orthographic and phonetic inconsistencies. This allows for phoneme-level Levenshtein distance to be used to determine pronunciation accuracy, which is more sensitive to substitution, insertion, and deletion of phonemes at the articulation level.

4. To offer interpretable pronunciation feedback: The system is designed to offer interpretable and actionable feedback to learners through a range of techniques,

including accuracy scores, word-level categorization, and IPA visualizations. This is to ensure that learners are offered detailed diagnostic information, as opposed to simple correct/incorrect feedback, to maximize pedagogical effectiveness.

Scope

The main objective of the proposed Automatic Pronunciation Mistake Detection System is the development of a multi-lingual, phoneme-level speech evaluation system, which can automatically identify pronunciation mistakes and provide structured feedback. The proposed system is expected to enable the development of a speech processing system based on machine learning, which can include Whisper-based ASR, grapheme to phoneme, DTW, and Levenshtein Distance-based phoneme comparison. It is expected to enable the development of a speech evaluation system, which can provide pronunciation accuracy scores and word-level feedback, making it useful in educational and self-learning scenarios while ensuring computational efficiency and scalability in a web-based framework.

Functionally, the system includes a browser-based user interface to support audio recording and sentence input, a backend processing engine to support speech transcription and phonetic analysis, and a structured scoring module to classify pronunciation quality at the word and phoneme level. It includes multilingual support for English, Hindi, and Marathi languages, as well as language-specific phonology support, such as schwa deletion, to support Devanagari script languages, and API-based modular integration to support extensibility in the proposed system design. The tangible outputs of this project include a working prototype web application, an ASR-based pronunciation evaluation pipeline, phoneme-level scoring algorithms, and the system architecture design.

The project is created under certain constraints, which include the availability of limited speech data in different languages. It is expected that there will be clear audio input from the users and standard quality input from the microphone to proceed with the transcription process and that the project will be complete once the end-to-end speech processing pipeline is integrated, the phoneme-level alignment and scoring are correct, and the feedback is generated in a structured format with consistent performance in all three languages under the evaluation benchmarks.

Related Work

1. Computer-Assisted Pronunciation Training(CAPT) System

The Computer Assisted Pronunciation Training tool is made to help people learn languages by checking how they say words and giving them feedback. This feedback is like a report that shows how they are doing.At first these tools were simple. They compared how people said words to how they should be said. They did this by using rules about how words sound.. These tools had some problems. They could not handle people who spoke differently or had accents. They also had trouble with people who spoke fast or slow.Now that speech technology is better the Computer Assisted Pronunciation Training systems are using methods to check how people say words. This makes the feedback more accurate and faster.

Recently some researchers found out that they can use machine learning to check how people say words. This means the computer can compare how a person says a word to how it should be said. A study by Patil and others in 2024 found that the computer can even find the mistake in a word by checking how similar it sounds to the correct word. The Computer Assisted Pronunciation Training tool is getting better at helping people learn languages by using these methods.

2. Automatic Speech Recognition for Pronunciation Assessment

Automatic Speech Recognition (ASR) technologies have really helped improve pronunciation assessment systems. Traditionally ASR systems were built using methods. These methods were used to develop ASR systems. They needed a lot of data and complex acoustic modeling to get accurate results. In years ASR has advanced quickly and can now recognize speech accurately using deep learning methods. These methods are also more accurate with self-supervised learning. A major development in this area is the wav2vec 2.0 model.

3. Grapheme-to-Phoneme Conversion

The role of grapheme-to-phoneme conversion is very important for pronunciation evaluation system, as it is used for converting written text into phonetic form for comparison with spoken audio. Earlier grapheme-to-phoneme conversion method were mostly based on rules and were implemented using pronunciation dictionaries. For English, there is a package called "eng_to_ipa," based on the CMU Pronouncing Dictionary, which gives a phonetic transcription of words. This is based on the

International Phonetic Alphabet and represents the expected pronunciation of words. One of these is Epitran, created by David R. Mortensen and colleagues. Epitran supports more than eighty languages and includes Unicode-based transliteration rules for converting written text into IPA. Epitran includes support for Indian languages like Hindi and Marathi under configurations like "hin-Deva" and "mar-Deva," respectively.

However, the Indic languages bring their own set of phonological challenge, which is not being addressed by the G2P models. For example, the process of schwa deletion is a phonological process that is observed in Hindi and Marathi, in which the inherent vowel sound 'ə' is deleted in the pronunciation of words. The G2P models may not be able to handle this process, which may affect the phoneme and pronunciation evaluation process, thus necessitating the need to incorporate additional linguistic post processing step.

4. Word Alignment and Sequence Matching

To figure out how well someone is pronouncing words we need to compare the words to the original text. We do this by looking at each word or sound to see how they match up. The common way to do this is by using something called the Levenshtein edit distance. This measures how changes we need to make to one sets of words to get the other sets. It is very useful for speech recognition and understanding language for the users.

Another way to compare the words to the original text is by using Dynamic Time Warping which we used in our project. This method helps us see how similar two things are when they happen over time like when we say words out loud. Dynamic Time Warping is used a lot in speech processing to compare spoken words to the words even if they are different lengths or happen at different times. This helps us evaluate how well someone is pronouncing words.

There are also complex ways to make sure the words line up correctly. For example we can use tools like Google OR Tools CP-SAT to make sure the words are aligned in the best way possible for us. This is like solving a puzzle where we want to find the match, between the spoken words and the original words. By doing this we can get an accurate idea of how well someone is pronouncing words in that language, which makes the evaluation more accurate.

5. Research Gap Summary

There has been lot of progress in figuring out people say words in English using computer programs that learn from examples. There are still some problems with systems that help people in practicing words in different languages. For example it is really hard to find a system that can help people practice saying words in English and Hindi and Marathi all at the exact time. These systems need to understand how these words sound in each language. The languages that use the Devanagari script like Hindi and Marathi have some sounds that are not found in English. These sounds are important for saying these words for Hindi and Marathi. The systems we have cannot do a good job in handling these special sounds.We need to make a system that can help people practice saying words and also understand how words sound in languages. This paper is about a system that can help people practice saying words, in English ,Hindi and Marathi. It pays attention to how words are really said in each language.

Problem Statement

Pronunciation is arguably the most neglected area in second language learning, not because its importance goes unrecognised, but because meaningful feedback is genuinely hard to deliver at any real scale. In a typical classroom a teacher managing thirty or more students simply cannot stop to give each learner word-by-word phonemic guidance within a single session. The feedback that does get given tends to be vague at best, and learners practicing alone have no reliable way of knowing whether what they produced actually matched the target. Errors that go unaddressed in the early stages of learning become habitual over time, and by the time a learner notices something is off, the habit is already quite difficult to break.

Existing speech technology does not solve this in any straightforward way. General purpose recognition systems are built to understand meaning, so they are deliberately tolerant of phonemic variation, which is the exact opposite of what pronunciation training actually needs. This difficulty compounds further for languages like Hindi and Marathi, where grapheme to phoneme tools carry their own inaccuracies. Epitran for instance appends word-final schwas that native speakers never vocalise, which creates a systematic gap between what the tool expects and what real speech contains. Real learner speech adds another layer of complexity altogether. Skipped words, hesitations and incomplete utterances mean that aligning what was said against what should have been said is itself a non-trivial problem, and systems that handle it poorly produce scores that mislead more than they help.

This project presents an end-to-end pronunciation assessment system operating at

the phoneme level across English, Hindi and Marathi. It brings together Whisper-based speech recognition with IPA-level edit distance scoring, DTW word alignment with a fuzzy fallback for disfluent input, and language-specific post-processing to correct known converter artefacts. Taken together, these components produce consistent per-word diagnostic feedback delivered through a standard browser, with no specialised hardware required and no human evaluator needing to be in the room.

Methodology

1. Overview

The system has a sequential pipeline where recorded audio passes through preprocessing, speech recognition, word alignment, phoneme conversion and accuracy scoring before result is returned to the user. Each component has been chosen specifically for multilingual pronunciation assessment rather than general speech processing and the overall design prioritises accessibility by running through a standard browser without requiring a specific hardware system.

2. Speech Data Acquisition and Preprocessing

Audio is recorded in the browser via the Web Audio API at 16,000 Hz, mono, 16-bit PCM matching native input format of the Whisper ASR model and thus, eliminates any need for resampling. Once the user stops recording, the audio is Base64-encoded and transmitted to the backend alongside reference sentence and language code. Before recognition two normalisation steps are applied: DC offset removal through mean subtraction and peak normalisation with a silence guard to prevent numerical errors on near-silent submissions.

3. Automatic Speech Recognition

Transcription is performed by OpenAI Whisper model via the Hugging Face transformers pipeline. Whisper was selected for its robustness with accent variation, native multilingual support and ability to operate without fine-tuning or forced alignment across English, Hindi and Marathi. The pipeline is configured to return the word-level timestamps, and the target language is enforced through generate_kwargs to ensure transcription rather than translation. Output consists of a transcript string and word-level start and end timestamps converted from seconds to sample indices for downstream use.

4. Reference Sentence and IPA Generation

Target sentences are drawn from semicolon-delimited CSV files, one per language, stored under ./databases/ folder. Difficulty is assigned at the runtime by word counts, with one to eight words classified as Easy, nine to twenty as Medium, and above twenty as Hard. The selected sentence is passed through the appropriate grapheme-to-phoneme converter to produce reference IPA strings. English uses eng_to_ipa, while Hindi and Marathi use Epitran with the hin-Deva and mar-Deva models respectively to produce the output. For both Indian languages, a post-processing step removes word-final schwas using the regular expression ə(?=\s|$), correcting a known Epitran artefact where a vowel that native speakers never vocalise is appended at word boundaries.

5. Word Alignment

The transcribed word sequence is aligned against the reference using Dynamic Time Warping on a word-level Levenshtein edit-distance matrix. The dtwalign library finds the minimum-cost monotone path through this matrix, mapping each reference word to its closest transcribed counterpart. When the DTW output leaves more than 40 percent of reference words unmatched a fuzzy fallback using difflib, SequenceMatcher takes over, pairing each reference word to the transcribed word with the highest similarity ratio above a threshold of 0.35. This fallback ensures reliable alignment on disfluent or heavily accented input where the primary path tends to fragment.

6. Phoneme Scoring and Feedback

Each aligned word pair is converted to IPA and compared using Levenshtein edit distance. Per-word accuracy is computed as:

$$Accuracy = max\left(0, \frac{N_{\Phi} - D_{edit}}{N_{\Phi}} \times 100\right)$$

Where,

$N_{\emptyset}$ is the reference phoneme count.

D_{edit} is the edit distance between the two IPA strings.

Sentence-level accuracy is the phoneme-weighted mean across all word pairs, in range of [0, 100]. Each word is then assigned a category. We can view it in Table 1.

Table 1: Pronunciation Accuracy Categories with Scoring Thresholds and Interpretations

Category	Threshold	Interpretation
Good (0)	≥ 80%	Phonemes closely matched the reference.
Okay (1)	60 - 79%	Minor Phonemes deviations present
Poor (2)	<60%	nificant Phonemes errors detected

System Requirements

1. Hardware Requirements

The system is designed to be deployed on a commodity hardware without dependency on any specialised hardware, making it viable for both laboratory and real-world instructional settings. The minimum and recommended configuration are as follows:

Table 2: Minimum and Recommended Hardware Requirements for System Deployment

Component	Minimum	Recommended
CPU	4-core, 2.0 GHz	, 3.0 GHz or faster
RAM	8 GB	16 GB
Storage	10 GB free	GB free (SSD preferred)
GPU	Not required	DIA GPU, ≥ 6 GB VRAM
Microphone	16-bit, 16 kHz mono	condenser microphone
Network	Broadband (Edge TTS only)	-latency broadband

All ASR inferences runs on CPU by default (device=-1 in the Whisper wrapper), ensuring compatibility across machine that lack dedicated graphics hardware. Thus, make it more adaptable. GPU acceleration is supported via PyTorch CUDA backend and is enabled by setting device=0 in the model configuration reducing Whisper-small inference time on a five-second utterance from approximately four seconds to under one second.

2. Software Requirements

The system is implemented in Python 3.9 or later. Core runtime dependencies are grouped by functional roles.

ASR and Phoneme Analysis

torch>=2.0 for tensor operation and neural network inference, transformers>=4.38 hosting the Hugging Face pipeline wrapping OpenAI Whisper, openai/whisper-small as the default pre-trained ASR backbone (auto-downloaded on first run, with whisper-medium available for higher-accuracy deployments), epitran >= 1.4 for rule-based grapheme-to-phoneme conversion for Hindi (hin-Deva) and Marathi (mar-Deva), eng_to_ipa for dictionary-based English grapheme-to-IPA conversion, dtwalign for Dynamic Time Warping alignment between estimated and reference word sequences, numpy >= 1.24 for numerical operations and edit-distance matrix construction, and difflib from the Python standard library for fuzzy string matching in the alignment fallback path.

Text-to-Speech

edge-tts provide Microsoft Edge neural TTS for Hindi and Marathi (voices: hi-IN-SwaraNeural, mr-IN-AarohiNeural, internet connection required), pydub handles MP3 to WAV conversion (requires the ffmpeg system binary), Silero TTS accessed via torch.hub provides local English TTS using the lj_16khz speaker with no

internet dependency and soundfile manages WAV file I/O

Web Application Layer
Flask serves as the HTTP server hosting the lambda-style endpoints, pandas >= 1.5 manages CSV-based sentence datasets, jQuery 1.7.1 and Chart.js serve as front-end dependencies delivered from CDN

Optional / Deployment
nest_asyncio is required when Edge TTS is invoked from within an already-running event loop. The ffmpeg system binary is required by pydub and must be installed, separately via the operating system package manager.

3. Data Requirements

Sentence datasets are stored as semicolon-delimited CSV files under ./databases/ folder, one file per supported language-data_en.csv for English, data_hi.csv for Hindi (Devanagari script) and data_mr.csv for Marathi (Devanagari script). Each file require at minimum a sentence column. No pre-labelled metadata beyond the sentence text itself is needed. At runtime, The software automatically classifies each sentence into one of three difficulty tiers by word count:

Table 3: Sentence Difficulty Tier Classification Based on Word Count

Tier	Word Count	Label
1	1–8 words	Easy
2	9–20 words	Medium
3	21+ words	Hard

4. Language Support

Table 4: Supported Languages with Corresponding ASR Codes and Phoneme Converters

Language	Script	ASR Code	Phoneme Converter
English	Latin	en	eng_to_ipa (dictionary)
Hindi	Devanagari	hi	Epitran hin-Deva
Marathi	Devanagari	mr	Epitran mar-Deva

For Hindi and Marathi, a post-processing schwa-deletion rule removes word-final /ə/ characters that Epitran appends but which are phonologically silent in natural connected speech. Without this correction, correctly pronounced words would incur an artificial edit distance penalty reducing scoring fidelity in these language.

System Architecture

1. System Architecture Overview

The proposed system has a five-layer client-server architecture. This architecture is made for assessing pronunciation in time and it supports many languages. It is designed in a way that each layer has its job and it talks to the layers next to it in a clear way. This makes the system easy to test and add languages to without having to change the parts that are already there. The main idea behind this system is that the important parts of the system do not depend on details. Instead they depend on ideas. For example the part of the system that scores pronunciation depends on interfaces like IASRModel and ITextToPhonemModel. This means that we can use any system we want to recognize speech or convert text to phonemes without having to change the rest of the system.
Let us look at how the system works. When a user speaks into their browser their speech goes through a series of steps. First the speech is encoded into a format. Then it is sent to a server using a method called REST. After that the speech is cleaned up. Prepared for analysis. Next a deep neural network is used to recognize the speech. The recognized speech is then converted into a format that can be compared to the pronunciation. The system then compares the user's pronunciation to the pronunciation and gives the user feedback. Each of these steps can be. Replaced without affecting the other steps. The system works the way, for English, Hindi and Marathi but it uses different components for some of the steps depending on the language.

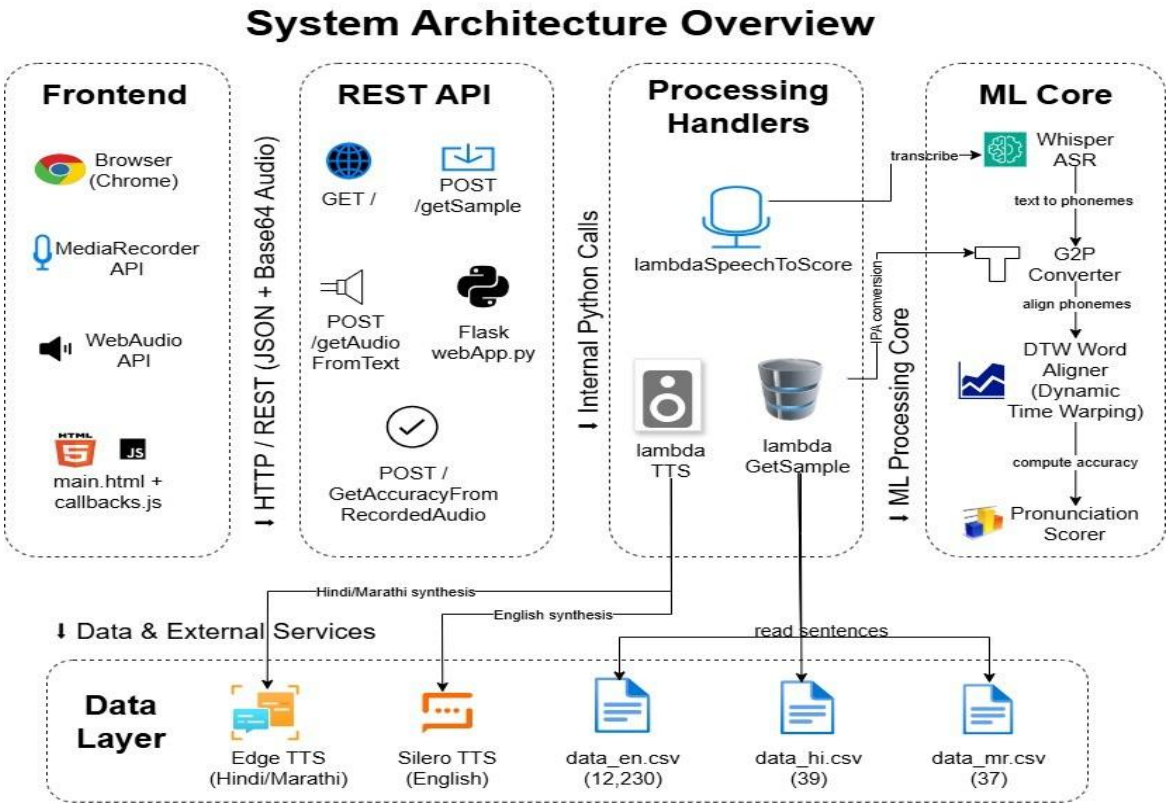


Fig 1: System Architecture

Layer 1: Frontend Interface

The frontend of the system is a website that users can access directly in their browser. It is made with HTML5, CSS and regular JavaScript. The website lets users do three things: make a practice sentence listen to how the sentence is supposed to sound and record themselves saying it.

When users record their voice the browser captures the audio using the MediaRecorder API. The recording is saved in OGG format, which's what Chrome and other similar browsers use, at a good quality of 48,000 Hz in stereo. Of sending the audio as a separate file the system changes the recording into Base64 format and sends it to the server with a JSON request. This way the process is easier. Users do not have to upload files.

After the system checks the pronunciation the results are shown in a way that's easy to understand. Each word in the sentence is highlighted with a color based on how it was pronounced:

- Green means the word was pronounced correctly which is eighty percent or above.
- Orange means it was partially correct which is between sixty and seventy nine percent.
- Red means it was pronounced incorrectly or skipped, which is, below sixty percent.

The system also has a section that shows the phonetic version of both the correct pronunciation and the users pronunciation. This helps people who are learning clearly see where they made mistakes and understand which sounds they need to work on. The phonetic version is shown in a format that experts use and it helps users see exactly what they did wrong. How they can improve the pronunciation of the practice sentence.

Layer 2: REST API Server (Flask)

The backend API is made using the Flask web framework. This framework has Flask-CORS enabled. This allows the backend API to handle requests from websites.The backend API has three endpoints. These endpoints are connected to the three things that users can do with the website.

text transcript of the spoken utterance, and (2) word-level timestamps expressed as sample indices at 16,000 Hz. These timestamps are later used to compute word-level intonation energy and to align recognised words with the reference text.

Endpoint	Method	Function
/getSample	POST	Returns a random practice sentence from the language-specific CSV corpus along with its IPA phoneme transcription, filtered by difficulty category.
/getAudioFromText	POST	Synthesises reference audio for the given sentence using the appropriate TTS engine (Edge TTS for Hindi/Marathi; Silero for English) and returns Base64-encoded WAV.
/GetAccuracyFromRecordedAudio	POST	Receives Base64-encoded audio and reference text; executes the full ASR → G2P → alignment → scoring pipeline; returns pronunciation accuracy and per-word feedback.

There are three modules that handle requests: lambdaGetSample, lambdaSpeechToScore and lambdaTTS. These modules are loaded only when they are needed. They are not loaded when the server starts up. This means that the big machine learning models, like Whisper and Silero TTS are not loaded until they are actually needed. This makes the website load faster and use memory when it is being developed. The machine learning models are only loaded when the lambdaGetSample, lambdaSpeechToScore and lambdaTTS modules are needed.

Layer 3: Processing Handlers

There are three handler modules that work between the HTTP API layer and the ML core engine. They take care of all the input and output tasks. This includes parsing requests decoding audio, managing files and formatting responses. This way the ML core engine does not have to deal with any HTTP tasks.

The lambdaSpeechToScore handler is the complicated one. It uses a strategy to load audio files so that it works in different environments without needing FFmpeg. It tries to load files in this order: soundfile, then librosa then pydub, then audioread. Finally it tries to parse the WAV file manually. If all these methods fail it uses Pythons built-in wave module to parse the WAV bytes. This makes sure that audio files can be loaded in any environment.

Layer 4: Machine Learning Core

The ML core is the technical heart of the system, comprising four sequential components that transform raw audio into a quantified pronunciation accuracy score. Each component is accessed through an abstract interface defined in ModelInterfaces.py, ensuring that concrete implementations can be swapped independently.

Automatic Speech Recognition — WhisperASRModel

Speech recognition is performed by OpenAI Whisper (whisper-small variant) loaded via the HuggingFace transformers pipeline with return_timestamps='word'. Whisper is a transformer-based encoder-decoder model pre-trained on 680,000 hours of multilingual speech, making it inherently capable of recognising Hindi and Marathi without task-specific fine-tuning. The force_language parameter constrains the model's language prior, substantially improving recognition accuracy for Indic languages by preventing code-switching to unintended languages.

The ASR output provides two critical data streams: (1) the full

Grapheme-to-Phoneme Conversion — RuleBasedModels

The reference text and the speech recognition transcript are changed into the International Phonetic Alphabet. This is done using tools that work with each language.

The function get_phonem_converter/language) gives us the tool based on the language we are using.

Here is how it works for some languages:

- English: We use a tool called EngPhonemConverter. This tool uses a library called eng_to_ipa. It also uses a dictionary from CMU to get the phonetic symbols for English words.
- Hindi: We use a tool called HindiIPA. This tool uses a set of rules from Epitran to change the Devanagari script into the International Phonetic Alphabet. Then we remove a sound called schwa.
- Marathi: We use a tool called MarathiIPA. This tool also uses Epitrans rules. Removes the schwa sound.

The schwa deletion is very important for getting the phonetic transcription. In Hindi and Marathi the script always includes a vowel sound after each consonant.. When people speak they usually do not say this vowel sound at the end of

words.
The tool Epitran does not know this. It keeps the schwa sound. To fix this we use a rule to remove the schwa sound when it comes before a space or, at the end of a word.
ipa = re.sub(r'ə(?=\s|$)', '', ipa)
For example the Hindi word नमस्ते is transcribed as /nʌmʌsteːə/ without this rule.. With the rule it is transcribed as /nʌmʌsteː/ which is how people really say it. If we do not use this rule the pronunciation will be wrong. This will give us incorrect results.

Word Alignment — Dynamic Time Warping
The Automatic Speech Recognition output is not very accurate. This means that words can be added, removed or replaced. So comparing the text and the Automatic Speech Recognition transcript word by word does not work well.
The system uses a method called Dynamic Time Warping over a matrix to find the best match between the two sets of words.
When we have M words that the system thinks it heard and N words that are actually in the text we create a big table. This table shows how similar each word is to every word. Each cell in the table has a number that shows how different the words are.
We add a row to the table to account for words that are in the original text but not in what the system thought it heard. The Dynamic Time Warping method then finds the way to match the words so each word in the original text is paired with the most similar word that the system thought it heard.
If more than 40% of the words in the original text cannot be matched the system knows that the Automatic Speech Recognition output is very poor. In this case it uses a method to find words that are similar but not exactly the same. This method tries to preserve the order of the words so it still makes sense.

Pronunciation Scoring
The way we figure out how accurate the pronunciation is is by looking at the phonemes. We use something called Levenshtein edit distance on the International Phonetic Alphabet strings for each pair of words that match up.For each pair of words the word and the word that was written down we get a score for how accurate the word is. This score is called the word accuracy.The word accuracy is calculated like this: we take the number of phonemes in the real word and we subtract the distance between the real word and the written word. Then we divide that by the number of phonemes in the real word and multiply by 100.

$$\text{Accuracy\_Word} = \left(\frac{|\text{real ipa}| - d(\text{real ipa}, \text{transcribed ipa})}{|\text{real\_ipa}|} \right) \times 100$$

The distance between the two words is like a measure of how different they're and it is calculated by looking at the phonemes one by one.

$$Accuracy\_Overall = \left(\frac{\sum |real\_ipa\_i| - \sum d\_i}{\sum |real\_ipa\_i|} \right) \times 100$$

The score is always between 0 and 100.
To get the accuracy of the sentence we add up all the phonemes in the real words and we add up all the distances between the real and written words. Then we calculate the score in the way as before.
We then put each word into a category based on how accurate it's

- If the word is really accurate with a score of 80 or more it is called Good.
- If the score is between 60 and 79 it is called Medium.
- If the score is, then 60 it is called Poor.

These categories are what drive the colours we use with Good being green, Medium being orange and Poor being red.

Layer 5: Data and External Services
The data layer comprises three CSV sentence corpora and two TTS services. Sentences are stored in semicolon-delimited CSV files with a single sentence column, encoded in UTF-8 to support Devanagari Unicode characters for Hindi and Marathi. Sentences are classified into three difficulty tiers by word count: Easy (1–8 words), Medium (9–20 words), and Hard (21+ words).
Reference audio is generated by a dual TTS strategy: Microsoft Edge TTS, using the Indian-accent neural voices hi-IN-SwaraNeural (Hindi) and mr-IN-AarohiNeural (Marathi) provides phonologically appropriate Indian pronunciation for Indic languages, while Silero TTS (lj_16khz speaker) operates fully offline for English. Using native Indian neural voices is linguistically important because Hindi and Marathi phonology includes retroflex consonants, aspirated stops, and nasalised vowels that are poorly rendered by generic Western TTS systems.

2. End-to-End Flow of Application

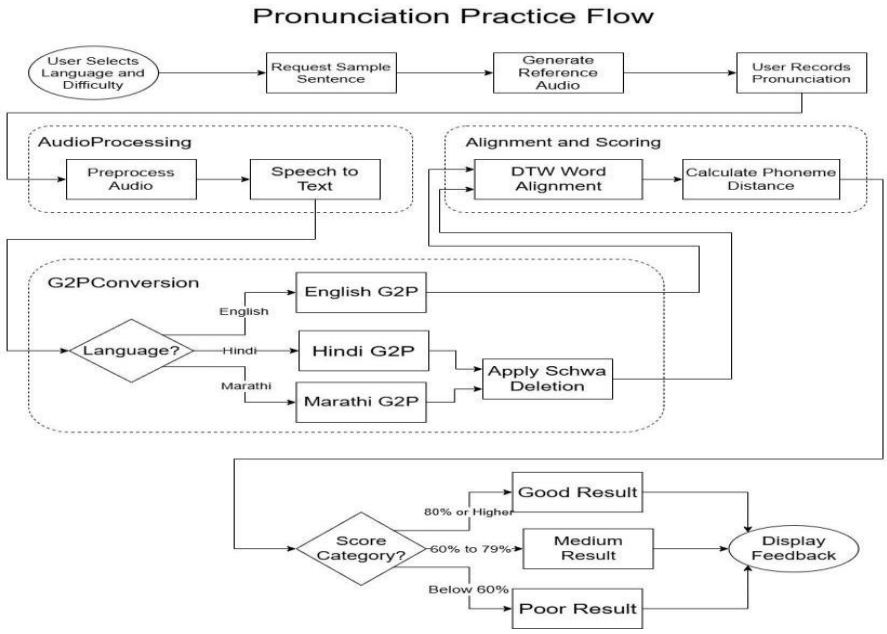


Fig 2: Pronunciation Practice Flow

The flow diagram shows how the pronunciation evaluation system works from start to finish. It shows how a users spoken words are turned into a score that shows how accurate they are.

The process starts when a user creates a sentence to practice and records themselves saying it on the website. The audio is then sent to the server as part of a request. This is where the user interaction stops and the computer processing starts.

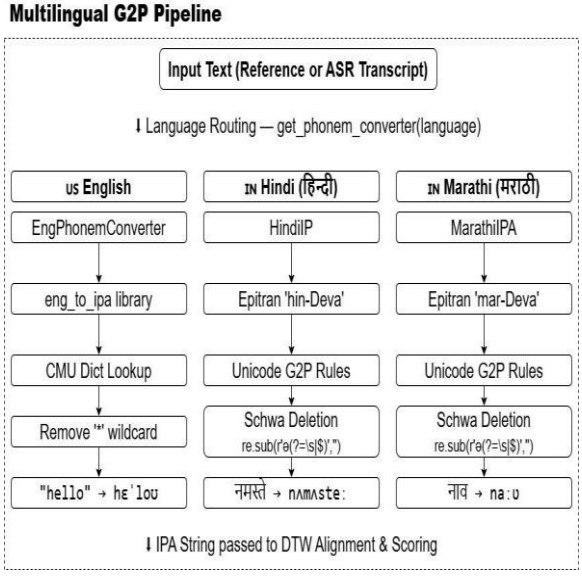
When the server gets the audio it starts processing it in stages. First it uses Automatic Speech Recognition (ASR) to turn the words into text. It also breaks down the text into words.

The original sentence and the text from the ASR are then converted into representations. This is done using a Grapheme-to-Phoneme (G2P) module. The phonetic representations are then aligned using Dynamic Time Warping (DTW). This helps to match the words with the original words. It also allows for words to be inserted, deleted or substituted.

The final stage is to calculate the pronunciation accuracy. This is done by comparing the phonemes of the aligned word pairs. A score is given for each word. These scores are added up to give an overall score for the sentence. The results are then sent back to the users browser. There they are shown as feedback.

The flow diagram shows how the system works in a step-, by-step way. It shows how the data is changed at each stage. The diagram goes from the users spoken words to the output.

3. Multilingual G2P Pipeline



The diagram shows the Grapheme-to-Phoneme processing pipeline. This pipeline changes text into a sound representation. The Grapheme-to-Phoneme processing pipeline works between the speech recognition output and the pronunciation scoring module. The main goal of the Grapheme-to-Phoneme processing pipeline is to make sure that the reference sentence and the recognized transcript are in the sound form using the International Phonetic Alphabet.

The Grapheme-to-Phoneme processing pipeline starts with choosing the language. The system picks the phoneme conversion model based on the language. For English the system uses a

dictionary to make sound symbols. For Hindi and Marathi the system uses rules to change the Devanagari script into symbols while keeping the natural sounds of the language. This way the system can apply language- sound rules without affecting other parts.

After making the sounds the Grapheme-to-Phoneme processing pipeline. Fixes the sounds. In languages the Devanagari script has a small vowel sound after consonants, which is often not said in spoken language. To make the sounds more real the system removes this vowel sound when it is not needed. This fix is important to prevent mistakes between the written sounds and the actual sounds.

The final output of the Grapheme-to-Phoneme processing pipeline is a sound sequence for each word. These sound sequences are used by the alignment and edit-distance scoring modules to compare sounds. By making the Grapheme-to-Phoneme processing pipeline a separate part the system ensures that the sounds are consistent the system is easy to extend. The pronunciation assessment is more reliable, for many languages.

4. Abstract Interface Layer

The main idea behind the architecture is to use Python Abstract Base Classes in ModelInterfaces.py. This is where we define the rules for each part of the system that can be changed. We have four interfaces.

Interface	Mandatory Methods	Used By
IASRModel	processAudio(), getTranscript(), getWordLocations()	WhisperASRModel, NeuralASR (legacy Silero)
ITextToPhonemModel	convertToPhonem(text) → str	HindiIPA, MarathiIPA, EngPhonemConverter, EpitranConverter
ITextToSpeechModel	getAudioFromSentence(str) → np.array	NeuralTTS (Silero English TTS)
ITranslationModel	translateSentence(str) → str	NeuralTranslator (Helsinki-NLP, legacy)

This layer helps to keep the pronunciation scoring pipeline separate from any Automatic Speech Recognition system. When we switched from the Silero CTC model to OpenAI Whisper we did not have to make any changes to pronunciationTrainer.py or WordMatching.py. We only had to add the WhisperASRModel class. The same thing applies to the Grapheme To Phoneme system. If we want to add support for an Indian language we just need to implement the ITextToPhoneModel interface with the right Epitran language code for that language. This makes it easy to add languages to the ModelInterfaces.py system specifically to the ITextToPhoneModel interface.

Results And Analysis

1. Speech Recognition Performance

Whisper-Small performed well on clean English speech with a Word Error Rate(WER) of around 6.25%. That said WER alone doesn't tell the whole story for pronunciation assessments. What matters more is whether the transcription captures enough of the phonemic details for scoring to be meaningful and Whisper-small handled that reasonably well across testing. For Hindi and Marathi performance depended quite heavily on how clear the speaker pronounced the words. Learners with weaker proficiency produced noticeably more transcription errors, which naturally affects the reliability of scores at the beginner level.

2. Processing Latency

Latency was measured on a CPU machine running an Intel Core i7-10700 with 16 GB RAM using five-second English recordings

Table 5: End-to-End Processing Latency Across Pipeline Stages

Processing Stage	Measured Duration
Audio preprocessing	Under 5 ms
Audio preprocessing	2.8 to 4.5 seconds
DTW word alignment	80 to 250 ms
IPA conversion and scoring	15 to 60 ms
Total end-to-end on CPU	Around 3 to 5 seconds

Most of that time sits in the transcription step. Everything else is relatively fast. The first request per language carries an additional 15 to 30 seconds as model loading cost, but after that the trainer is cached and stays loaded for the rest of the session, reducing time in later.

3. Word Alignment

DTW(Dynamic Time Warping) alignment

worked reliably on clean recording. The issues came up with disfluent input, specifically skipped words, heavy accents, and incomplete utterance, where the DTW path would sometimes break down. The fuzzy fallback using SequenceMatcher addressed this. Before it was added, the system would occasionally return scores with no real relationship to what was said. After it was added, that problem did not occur again across the inputs that were tested.

4. Pronunciation Scoring

Results followed a distribution that made intuitive sense. Speakers with strong proficiency in the target language scored mostly in Good range (above 80%) at the sentence level. Intermediate speakers fell around the Okay band. Beginners, especially English speakers working through Hindi or Marathi sounds that do not exist in English, frequently scored in the Poor category on harder words. This is not really a problem with the system. It is the system doing its job correctly.

5. Schwa Deletion Fix

The schwa deletion post-processing step had a more visible effect than initially anticipated. Without it, words at sentence boundaries in Hindi and Marathi were routinely penalised even when the speaker had pronounced them correctly. Epitran was adding a "/ə/" at end of those words that no native speaker would have actually said, and the scoring system was treating that as error which was incorrect. Removing it through the regular expression ə(?=\s|$) fixed those false penalties. Correctly pronounced boundary words moved into the Good category consistently after the fix was applied.

6. Text-to-Speech Quality

Silero produced clean English audio at 16 kHz. The Edge TTS voices for Hindi and Marathi were noticeably more natural sounding, with hi-IN-SwaraNeural and mr-IN-AarohiNeural producing correct results on all sentences tested. The per-word playback feature, where learners can listen to the reference and their own recording side by side for a single word, worked as intended and gave learners a straightforward way to hear exactly where they went wrong.

7. Dashboard

The dashboard reads word level history from Local Storage and displays total attempt, average accuracy, category breakdown, and score trend chart across the 30 most recent sentences. The export and import functions worked correctly across devices in all cases tested.

8. Limitations

Four limitations of the current implementation warrant acknowledgment.

- ASR Accent Bias - Whisper demonstrates superior recognition performance for American English compared to Indian accent, and overall native English accents demonstrate higher accuracy than non-native accent due to model training dataset. This introduces a systematic advantage for learners whose L1 phonology is closer to the model's dominant training distribution.
- Suprasegmental Features are absent from scoring - The evaluation pipeline assesses phoneme-level accuracy exclusively. Other dimension including pitch, rhythm, sentence stress, and connected speech phenomena are not quantified, despite their established contribution to perceived intelligibility and oral proficiency.
- Static Scoring Thresholds - The Good, Okay, Poor categories boundaries are fixed for all user and languages, with no mechanism for adaptation to learner's baseline proficiency.
- Internet Dependency for Hindi and Marathi TTS - For these two languages reliance on Microsoft Edge TTS, cloud-hosted service which is unavailable in offline deployment which is a limitation for users with no internet. English TTS runs entirely locally via Silero and carries no such constraint.
- Sensitivity to ASR-Induced Alignment Errors: Pronunciation assessment pipelines rely significantly on the quality of ASR transcription output. ASR recognition errors, such as word insertions, deletions, or substitutions, can have a "domino effect" on the subsequent stages of the evaluation pipeline. In cases where direct word positional comparison are performed between the recognized and the reference transcripts, alignment errors can result in a "domino effect" of evaluation errors. For example, if one word is missed by the ASR system, the subsequent word comparisons can be shifted, resulting in the evaluation of correctly pronounced words being reported as error. Many existing pronunciation assessment system have not incorporated specific word sequence alignment strategies to account for ASR recognition errors. This problem becomes more critical, especially for spontaneous speech or multilingual scenarios, where the ASR system might have varying recognition capabilities for different languages.

Future Scope

Despite the proposed pronunciation detection

system showing promising results for pronunciation evaluation across different languages, there are various modifications and improvements that can be made in future versions of the proposed pronunciation detection system for better performance and accuracy.

1. Expansion of Indic Language Corpora:

The pronunciation detection system can be made better in the future. One way to do this is to add Hindi and Marathi sentences to the system. The system uses these sentences to check how well people pronounce words. Now the system does not have as many Hindi and Marathi sentences as it has English sentences. If we add sentences it will be easier to check pronunciation. We should try to get least 5,000 Hindi and Marathi sentences. This will give us a lot of sentences to check pronunciation. To get all these sentences we can use things that already exist, like IndicTTS and Mozilla Common Voice Indic. These can help us make a collection of Hindi and Marathi sentences. The pronunciation detection system will be better, with Hindi and Marathi sentences..

2. Better Modeling of Schwa Deletion:

One more thing that needs to be improved is how we model the rules for removing sounds in Hindi and Marathi languages. Now we have a basic rule that removes the schwa sound at the end of words. But the thing is, removing sounds is actually a lot more complicated in these languages. It has to do with how consonants are grouped together and how words are structured. To make it better we could work on adding detailed rules for removing schwa sounds, based on the grammar rules of Panini. This would help our system to convert written words into sounds more accurately. We need to focus on deletion rules, for Hindi and Marathi and make sure our system follows these rules to produce better results.

3. Prosodic Feature Analysis:

The system looks at how words are said at the basic sound level. It does this now.. People can do more work later to make the system look at more than just the sounds. This means it can also look at things like how high or low the voice's which sounds are emphasized and the rhythm of the words. We can look at how the voice goes down when people talk and how loud or soft they are. This can help people know if they are saying words in the way with the right intonation patterns of words. The system can give people information, about the intonation patterns of words.

4. Phoneme Confusion Analysis for Targeted Feedback:

Another extension of the current system can include the The development of a phoneme confusion matrix is an idea. This matrix can help us figure out which phonemes people get mixed up. For instance people often get confused between retroflex and dental consonants and aspirated and unaspirated consonants.If we make this matrix the system can give users feedback about the phonemes they are confusing. This will be a feature that our system can offer and it will help users understand where they are going wrong with phoneme confusion.These new features can really help make our system better at giving users feedback. The phoneme confusion matrix and the feedback it provides will be a help, to users who are trying to improve their pronunciation of phonemes.

References

Zhang L, Zhao Z, Ma C, Shan L, Sun H, Jiang L, Deng S, Gao C. End-to-End Automatic Pronunciation Error Detection Based on Improved Hybrid CTC/Attention Architecture. Sensors. 2020; 20(7):1809.

Automatic Pronunciation Mistake Detector K. Sanath A1 , Vignesh Chandra 2 , Mrs. V. Veena 3 1,2Student, 3Guide, Asst. Professor, IT Department of Information Technology, Mahatma Gandhi Institute of Technology (A) DOI

Automatic Pronunciation Mistake Detector Vinita Vikram Patil, Pratiksha Mohan Wadkar, Aishwarya Bhagwan Mohite, Ms.

Priyanka Rajendra Jadhav, AUTOMATIC PRONUNCIATION VERIFICATION FOR SPEECH RECOGNITION, Kanishka Rao, Fuchun Peng, Françoise Beaufays, Google Inc., USA

Yingming Gao, Hai Shuang, Xiaoli Feng, Jingwen Cheng, Linkai Peng, Ya Li, Jinsong Zhang, and Min Liu. 2025. A Preliminary Study on Automatic Pronunciation Error Detection for Hearing-impaired Children. In Proceedings of the 2024 10th International Conference on Communication and Information Processing (ICCIP '24). Association for Computing Machinery, New York, NY, USA, 632–637.

Automatic Pronunciation Mistake Detector Aditi Tiwari, Abdul Sameer, Dr. Gousiya Begum Department of CSE, MGIT (A),

Gandipet ,Hyderabad ,500075 ,Telangana ,India

Automatic Pronunciation Assessment using Self-

Supervised Speech Representation Learning Eesung Kim, Jae-Jin Jeon, Hyeji Seo, Hoon Kim AI Lab, Kakao Enterprise.

Open AI, “Whisper: Robust Speech Recognition via large-scale weak Supervision,” Opne AI research 2022.

Y. El Kheir, A. Ali, and S. A. Chowdhury, “Automatic Pronunciation Assessment: A Review,” Findings of the Association for Computational Linguistics (EMNLP), 2023

W.-N. Hsu, B. Bolte, Y. H. Tsai, K. Lakhotia, R. Salakhutdinov, and A. Mohamed, “HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units,” IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2021.

E. Kim, J. Jeon, H. Seo, and H. Kim, “Automatic Pronunciation Assessment Using Self-Supervised Speech Representation Learning,” Proceedings of Interspeech, 2022.

D. Cai, “Developing an Automatic Pronunciation Scorer: Aligning Acoustic and Linguistic Features,” Language Learning Journal, 2025.

R. C. C. Shekar, M. Yang, K. Hirschi, and J. H. L. Hansen, “Assessment of Non-Native Speech Intelligibility Using Wav2vec2-Based Mispronunciation Detection and Multi-Level Goodness of Pronunciation Transformer,” Proceedings of Interspeech, 2023.

D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed., Pearson, 2023.

A. Gulati et al., “Conformer: Convolution-Augmented Transformer for Speech Recognition,” Proceedings of Interspeech, 2020.

D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed., Pearson, 2023.

Attapol Rutherford, Fuchun Peng, and Franc¸ois Beaufays, “Pronunciation learning for named-entities through crowdsourcing,” in Proceedings of Interspeech, 2014

Chun, Dorothy M. "Computer-Assisted Pronunciation Teaching: The State of the Art." Studies in Second Language Acquisition 30, no. 3(2008): 433–465.

Eskenazi, Maxine. "Using Automatic Speech Processing for Pronunciation Feedback in Second Language Learning." Language Learning&Technology 3, no. 2 (1999): 62–76

Hardison, Debra M. "The Role of Technology in Pronunciation Training." Journal of Second Language Pronunciation 1, no. 1 (2015): 29–56

Vinita Vikram Patil, Pratiksha Mohan, Aishwarya Bhagwan Mohite, “Automatic Pronunciation Mistake Detector”, International Journal of Creative Research

B.-C. Yan, M.-C. Wu, H.-T. Hung, and B. Chen, “An end-to-end mispronunciation detection system for l2 english speech leveraging novel anti-phone modeling,” INTERSPEECH, 2020

Hu, W.P.; Qian, Y.; Soong, F.K.; Wang, Y. Improved mispronunciation detection with deep neural network trained acoustic models and transfer learning based logistic regression classifiers. Speech Commun. 2015, 67, 154–166

Xiao Li, Asela Gunawardana, and Alex Acero, “Adapting grapheme-to-phoneme conversion for name recognition,” in Proceedings of ASRU, 2007