Journal of Information Systems Engineering and Management
2025, 10(46s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Real-Time Multilingual Speech Recognition and Language Mapping for Indian Code-Switched Speech

Mayur M. Jani[1], Sandip R. Panchal[2], Hemant H. Patel[3]
[1]*Research Scholar, Department of Computer Engineering, Dr. Subhash University, Junagadh, India.*
[2]*Professor, Department of Electronics and Communication Engineering, Dr. Subhash University, Junagadh, India.*
[3]*Associate Professor, Department of Computer Science and Engineering, Dr. Subhash University, Junagadh, India.*

ARTICLEINFO

Received: 30 Dec 2024

Revised: 12 Feb 2025

Accepted: 26 Feb 2025

ABSTRACT

Introduction: In a country like India, where people often switch between languages in everyday conversations, recognizing such mixed-language speech—also called code-switching—is especially tough for standard speech recognition systems. The challenge becomes even greater when the spoken input includes less-resourced Indian languages or uses Roman script.

Objectives: This work introduces a real-time speech recognition system designed to handle multilingual input—specifically Hindi, Gujarati, and English—without requiring users to choose a language beforehand. The main aim is to simplify the transcription of mixed-language speech while ensuring the output appears in the correct script for each word.

Methods: The system listens continuously and identifies the language of each word on the fly. It uses bilingual dictionaries to convert Romanized and code-switched words into their proper script forms. The interface is built like a simple Notepad, using Python and Tkinter, and relies on the Google Speech API for transcription. Users can not only transcribe but also save or share the output easily.

Results: Tests show that the tool performs well across a range of sentence types, even when the structure is complex or languages change mid-sentence. It achieves high accuracy in both speech recognition and script conversion, with minimal delay.

Conclusions: By combining real-time processing, automatic transliteration, and an easy-to-use interface, this system fills a crucial gap left by current ASR solutions. It offers a practical way for people in multilingual communities to document, communicate, and share spoken content more effectively.

Keywords: Multilingual Speech Recognition, Code-Switching, Indian Languages, ASR, Language Identification, Real-Time Transcription, Speech-to-Text.

INTRODUCTION

India, with over 22 official languages and hundreds of dialects, represents a uniquely complex linguistic landscape that poses major challenges for Automatic Speech Recognition (ASR) systems [21, 30, 34, 40]. Daily conversations among bilingual and trilingual speakers often include spontaneous code-switching—seamless transitions between two or more languages within a single sentence or phrase [13, 19, 36,41]. This linguistic phenomenon is prevalent in educational environments [30], workplaces [35], media [7], and online platforms [40].

Although commercial ASR tools such as Google Speech-to-Text [15], Microsoft Azure, and Whisper [17, 37] offer multilingual capabilities, they typically require prior language selection and perform poorly when confronted with code-switched speech, particularly for low-resource Indian languages [2, 4, 5, 9, 11]. Research in bilingual end-to-end speech recognition with universal decoding [16] has attempted to address these challenges, but practical real-world deployments remain limited. Additionally, these systems struggle with Romanized inputs and lack robust mappings to scripts like Devanagari or Gujarati, limiting their effectiveness in real-world use [10, 18, 31, 39].

Academic research has introduced various innovations, including federated learning-based models (CodeFed) [4], adaptive normalization (AdaCS) [9], dual script recognition [29], and Whisper-driven multilingual real-time transcription [17,37]. Moreover, benchmark datasets like MECOS [2], VITB-HEBiC [5], CS-Dialogue [24], CoSTA [23], and PIER [8] have laid foundational work in evaluating multilingual ASR performance. However, most studies emphasize backend improvements and overlook user-facing applications, often requiring substantial computational resources and machine learning expertise [6,25,31,33].

Emerging studies highlight the potential of low-resource multilingual ASR through techniques like efficient model compression [20], zero-shot recognition [3,22], and semi-supervised training [33,34]. Nevertheless, real-time deployment remains limited.

To bridge this research-practice gap, we propose a real-time Multilingual Speech Notepad, tailored to the Indian linguistic ecosystem. Our system integrates ASR with dynamic word-level language detection, automatic mapping of Romanized Hindi and Gujarati words to their respective Unicode scripts [27,32,39], and a GUI-driven interface to transcribe, edit, save, or share multilingual content [15,35]. This accessible solution brings advanced speech recognition to non-expert users—students, educators, and professionals—working in multilingual environments.

LITERATURE REVIEW

Multilingual ASR and Code-Switching Corpora

The development of reliable ASR systems depends heavily on high-quality datasets. Initiatives such as MECOS [2], designed for Manipuri-English code-switching, and VITB-HEBiC [5], focusing on Hindi-English, have contributed significantly to data collection for low-resource Indian languages. Similarly, CS-Dialogue [24], CoSTA [23], and PIER [8] aim to establish standardized evaluation benchmarks for code-switching (CS) ASR systems across various language pairs. Datasets like SEAME [7] and BANGOR-MIAMI [41] have also facilitated research in bilingual and multilingual code-switching contexts. However, these corpora remain largely academic and are not integrated into real-time, user-accessible tools.

Modeling Innovations for Low-Resource Languages

Various architectural and training approaches have been proposed to improve ASR for underrepresented languages. CodeFed [4] utilizes federated learning for distributed model training, while AdaCS [9] adapts normalization techniques for fluctuating language conditions. Other models incorporate Transformer-Transducer frameworks [28], MLP-LSTM hybrids [12], and dual-script E2E models [29]. Innovations like Attention-guided adaptation [27], interactive language bias [26], semi-supervised learning [34], and tokenizer unification [25] have enhanced CS-ASR performance, though many of these are computationally intensive and not readily deployable. Lightweight ASR models for resource-constrained environments [20,33] offer promising directions but are still rarely found in live applications.

Language Identification and Script Mapping

Effective transcription of CS speech hinges on precise word-level language identification and script rendering. Approaches such as kNN-CTC for zero-shot recognition [31], zero-resource and zero-shot benchmarks [3,22], and large language model filtering [6,33] show promise in identifying and handling unknown code-switches. Techniques for bilingual mapping, Romanized-to-Unicode conversion, and fuzzy dictionary alignment [27,32,39] have been explored, though few are realized in consumer-facing applications. Studies on phonotactic models and code-switch prediction [11,40] further highlight the complexity of multilingual speech recognition in spontaneous settings.

User-Centric Interfaces and Real-Time Tools

While tools like Whisper have facilitated real-time transcription [17,37], they often operate in fixed-language modes. Solutions such as Multilingual Transcription in Conferencing Settings [1,14], speech-to-text systems for low-end devices [35], and Google Cloud API implementations [15] have attempted to integrate ASR into accessible platforms. Yet, features like native script mapping, save/share options, and GUI-based interaction remain scarce. Applications like emotion-aware ASR [38], cross-lingual ASR using embeddings [36], and video conference transcription [14] offer niche functionalities, but lack holistic, multilingual adaptability.

Speech Emotion and Educational Applications

Some ASR models have expanded beyond speech transcription to include emotion recognition [32,38], language learning tools [12], and support for educational settings [30]. Systems like those presented in [40,41] investigate educational scenarios involving multilingual and code-switched inputs. Such integrations highlight the interdisciplinary potential of ASR but also underscore the need for unified, usable systems tailored to real-world Indian use cases.

Research Gaps and Motivation

Across the body of work reviewed, several critical gaps persist:

- Real-time ASR tools that detect language at the word level in spontaneous speech remain rare [22, 25, 31, 41].
- Most systems do not convert Romanized Indian language inputs into native scripts dynamically [10, 18, 27, 29, 39].
- Few tools offer a complete GUI-based interface for live transcription, editing, saving, and sharing multilingual content [15, 17, 35].

Thus, despite rich academic contributions, practical, deployable ASR solutions for multilingual Indian users remain underdeveloped. Our work addresses these gaps through a lightweight, script-aware speech Notepad that combines robust multilingual ASR with usability, enabling real-world deployment across educational and professional domains.

PROPOSED METHODOLOGY

The proposed system is a real-time, multilingual speech recognition application built to handle spontaneous code-switching between Hindi, Gujarati, and English. It offers a practical solution for Indian users by integrating speech capture, language detection, script mapping, and a user-friendly text editor into one cohesive platform.

System Overview

The architecture is implemented as a desktop application using Python, featuring a GUI developed with Tkinter and speech processing powered by the Google Web Speech API. It listens for spoken input, converts it to text, detects each word's language, and maps recognized Romanized words into their respective Unicode forms using editable CSV-based bilingual dictionaries. The overall architecture consists of the following key components:

- **Speech Input via Microphone**: Real-time audio is captured through the system's microphone.
- **Automatic Speech Recognition (ASR)**: Speech is converted to text using the Google Web Speech API, accessed via Python’s SpeechRecognition library.
- **Word Mapping Engine**: Recognized English words are checked against pre-defined Gujarati and Hindi word mappings stored in CSV files.
- **Dynamic Word Replacement**: When a word is found in the Hindi or Gujarati dictionary, it is replaced with its corresponding Unicode representation.
- **Text Display in Notepad**: The final multilingual text (containing English, Hindi, and Gujarati words) is displayed in a Tkinter-based text editor, simulating a Notepad interface.

The architecture allows users to speak in a mix of languages, and the system seamlessly handles language switching at the word level, without requiring manual language selection.

Overall Workflow

The following **Figure 1** illustrates the overall workflow of the proposed system, showing the sequence of steps from speech capture to multilingual text display.

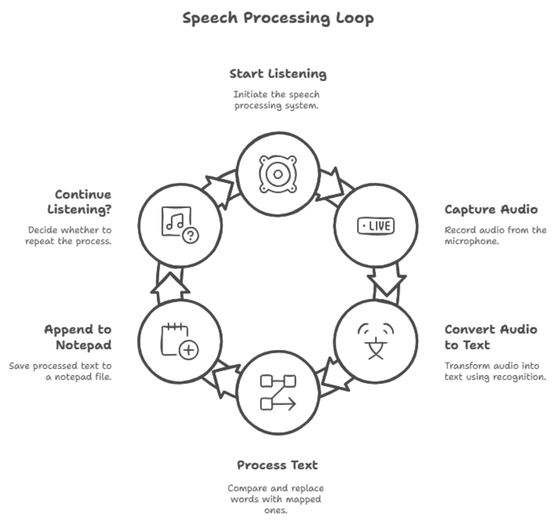

Speech Processing Loop
Start Listening
Initiate the speech processing system.
Capture Audio
Record audio from the microphone.
• LIVE
Convert Audio to Text
Transform audio into text using recognition.
Process Text
Compare and replace words with mapped ones.
Append to Notepad
Save processed text to a notepad file.
Continue Listening?
Decide whether to repeat the process.

Figure 1: Overall Workflow Diagram of the Proposed System

Step 1: Speech Capture from Microphone

- The microphone is activated when the user selects the “Start” option from the File menu.
- Continuous real-time audio capture is handled using Python’s speech_recognition library.
- To improve recognition accuracy, the system performs ambient noise adjustment using the adjust_for_ambient_noise() method.

Step 2: Speech Recognition (ASR)

- Captured audio is sent to the Google Web Speech API, which returns recognized text.
- The recognized text is in plain English (including Romanized Hindi or Gujarati words if spoken that way).
- Example:
    - **Spoken**: "Hello दोस्तों કેમ છો आज मौसम अच्छा છે"
    - **Recognized**: "Hello dosto kem cho aaj mausam accha che"

Step 3: Word-by-Word Language Mapping

- The recognized text is split into individual words.
- Each word is compared against two pre-loaded CSV dictionaries:
    - Gujarati mappings (from gu_data_cleaned.csv)
    - Hindi mappings (from hi_data_cleaned.csv)
- If a word exists in either dictionary, it is replaced with its mapped Gujarati or Hindi Unicode word.
- If no mapping is found, the word is retained in English.
- This hybrid word-level processing allows natural code-switching between languages without requiring the user to pre-select a language.

Table 1: Example Processing

Detected Word	**Gujarati Mapping**	**Hindi Mapping**	**Final Output**
Hello			Hello
dosto		दोस्तों	दोस्तों
kem	કેમ		કેમ

cho	છો		છો
aaj		आज	आज
mausam		मौसम	मौसम
accha		अच्छा	अच्छा
che	છે		છે

Table 1 illustrates the word mapping process, showing how each recognized word is compared against the Gujarati and Hindi dictionaries, with matching words being replaced by their respective Unicode equivalents, while unmatched words are retained in English.

Final Output: Hello दोस्तों કેમ છો आज मौसम अच्छा છે.

Step 4: Real-Time Display in Notepad

The processed text (containing a mix of English, Hindi, and Gujarati) is immediately displayed in the Tkinter text widget. Each processed segment (corresponding to one recognized speech block) is appended with a newline (\n) to ensure clarity and readability. This ensures that each speech input block appears on a new line in the Notepad interface.

Algorithm Pseudo Code

```
Start Application
Load Hindi and Gujarati CSV dictionaries
Initialize GUI with menu: Start, Stop, Save, Share, Export PDF
On "Start":
  Begin background audio capture
  On detecting stop word:
    Convert speech to text
    For each word:
      If in Gujarati CSV → map to Gujarati
      Else if in Hindi CSV → map to Hindi
      Else → keep as English
    Display final sentence in editor
On "Save"/"Share"/"Export PDF":
  Perform corresponding action
On "Exit":
 Close application
```

IMPLEMENTATION AND EXPERIMENTAL SETUP

This section outlines the technical environment, tools, functionalities, and performance metrics used in developing the proposed Multilingual Speech Notepad. The system is designed as a standalone desktop application that captures real-time speech, supports code-switched language recognition (English, Hindi, and Gujarati), and allows user interaction through a menu-driven interface offering multiple utilities like save, share, and PDF download.

Development Environment

Table 2: Software libraries and development tools

Component	Technology/ Library	Purpose
GUI Framework	Tkinter	Notepad interface for text display and controls
Speech Processing	SpeechRecognition	Capturing real-time speech from microphone
Speech-to-Text	Google Web Speech API	Converting speech to text
Data Handling	CSV Files	Storage format for the bilingual word mappings for Hindi and Gujarati.
Language Mapping	pandas	For efficient loading, searching, and mapping of Romanized words to Unicode representations using pre-defined CSV files.
Audio Input	PyAudio	Capturing audio stream from system microphone
Tokenization	re (regex)	For efficient tokenization of recognized text into individual words, allowing accurate language mapping.
PDF Generation	FPDF	To export final content as a PDF
Sharing Integration	Webbrowser / mailto, WhatsApp API	For sharing transcribed notes via Gmail and WhatsApp Web

The proposed Multilingual Speech Notepad was developed as a standalone desktop application, designed to operate in real-time for capturing multilingual code-switched speech in English, Hindi, and Gujarati. Python was chosen as the primary development language due to its extensive support for speech processing libraries, GUI development frameworks, and data handling capabilities. Table 2 shows the utilizedsoftware libraries and development tools.

Experimental Workflow and Application Flow

The user interacts with the application through a menu-driven interface, where each option triggers a core function of the system. The user first selects the "Start" option, which activates the microphone and begins continuous real-time speech recognition using the Google Web Speech API. The recognized text, which may include English, Romanized Hindi, and Romanized Gujarati, is then processed word-by-word. Each word is matched against pre-defined CSV dictionaries for Hindi and Gujarati. Words that match are replaced with their Unicode equivalents; unmatched words are assumed to be English and retained as-is.

Figure 2 Experimental Workflow of the Menu-driven Application, illustrating the sequence from microphone activation and real-time speech recognition to multilingual word mapping, text display, and extended features like Save, Download as PDF, Share, and Exit.

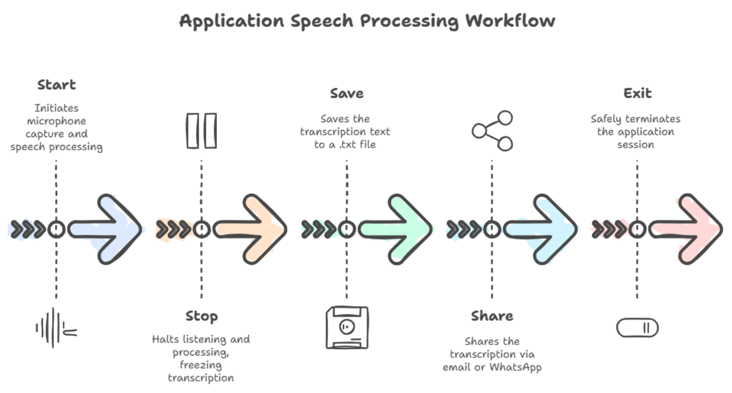

Application Speech Processing Workflow
Start
Initiates microphone capture and speech processing
Stop
Halts listening and processing, freezing transcription
Save
Saves the transcription text to a .txt file
Share
Shares the transcription via email or WhatsApp
Exit
Safely terminates the application session

Figure 2: Experimental Workflow of Menu-driven Application

When the user selects "Stop", the speech recognition process ends, and the captured content can be manually reviewed or edited. The application also supports several extended features:

- **Save:** Stores the captured text into a local .txt file. Download as PDF: Exports the multilingual note into a formatted .pdf document.
- **Share:** Offers direct sharing via Gmail or WhatsApp Web using URL schemes or browser APIs.
- **Exit:** Cleanly shuts down the application and releases all resources.

These features improve usability by supporting storage, portability, and real-time communication of multilingual notes.

Performance Metrics

The system's performance was assessed using the following key metrics:

- **Recognition Accuracy (ArA_rAr)**

Defined as the percentage of words correctly recognized (before mapping).

$$A_r = \frac{N_{correct}}{N_{total}} \times 100$$

- **Mapping Accuracy (AmA_mAm)**

Defined as the percentage of recognized words correctly mapped to the correct Unicode representation.

$$A_m = \frac{N_{mapped\_correctly}}{N_{mappable}} \times 100$$

- **Processing Latency (TpT_pTp)**

Average time from audio capture to display in the Notepad (measured in milliseconds).

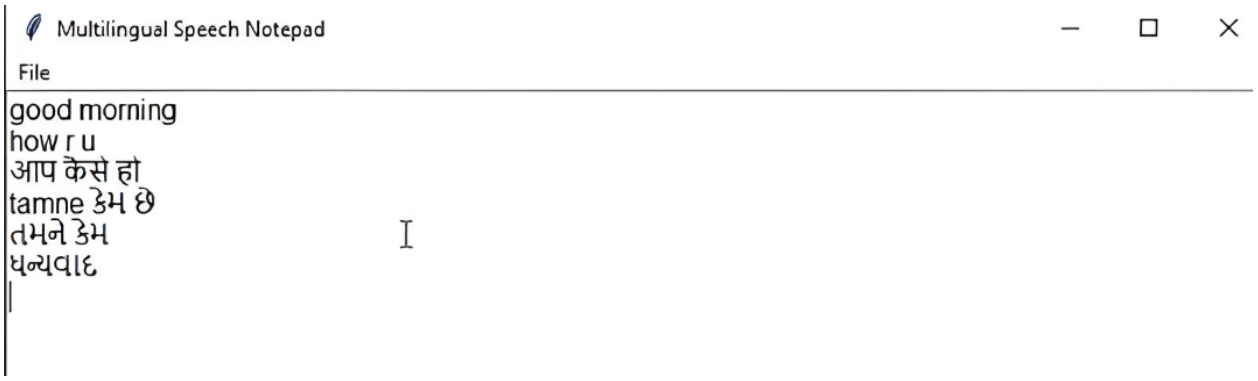
$$T_p = \frac{\sum_{i=1}^{n} t_i}{n}$$

Where ti=1 to ti=n is the processing time for eachcaptured segment.

Application Interface Snapshots

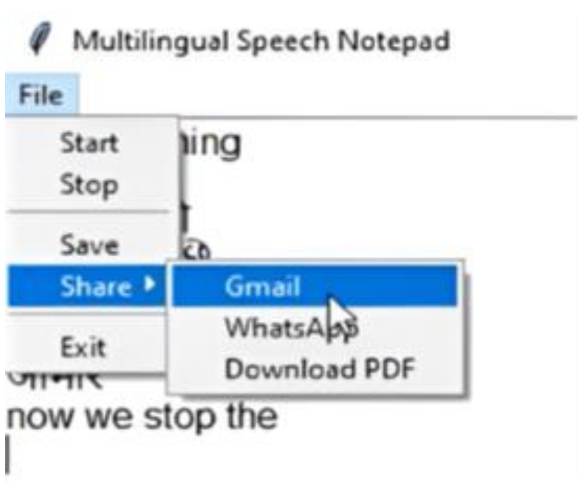
To provide a clear visualization of the user interface and functional flow of the developed Multilingual Speech Notepad, a series of snapshots are presented below. These images demonstrate the key stages of user interaction within the application. **Figure 3** captures the live transcription phase, showcasing how a code-switched sentence is recognized and displayed in real time within the notepad. **Figure 4** illustrates the initial user interface, highlighting the "File" menu with options such as Start, Stop, Save, Share, and Exit. **Figure 5** presents the post-processing interface, where users can save their transcription as a .txt file or share it via Gmail, WhatsApp, or download it as a PDF.

Journal of Information Systems Engineering and Management
2025, 10(46s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/
Research Article



good morning
how r u
आप कैसे हो
tamne કેમ છે
તમને કેમ
ધન્યવાદ

Figure 3: Live transcription of a code-switched sentence



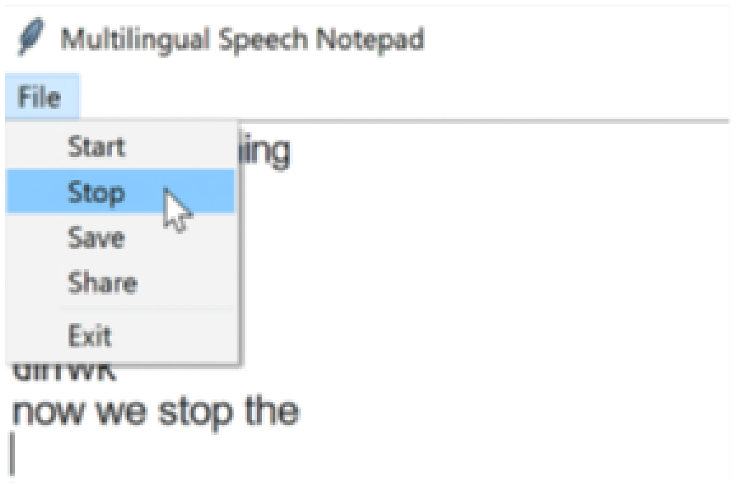


Figure 4: Initial UI with File Menu **Figure 5:** Output saved and shared via PDF/ Gmail/ WhatsApp

RESULTS AND DISCUSSION

This section presents an in-depth evaluation of the proposed Multilingual Speech Notepad, focusing on system performance, mapping effectiveness, and user interaction. The results were obtained through rigorous testing using controlled multilingual inputs and real-time speech scenarios in Hindi, Gujarati, and English. Emphasis is laid on code-switching efficiency, mapping fidelity, and latency control.

Recognition Accuracy

The speech recognition module employs the Google Web Speech API to transcribe real-time multilingual utterances. Recognition accuracy was measured across multiple trials involving code-switched sentences.

For example,

The following test utterance:

"Hello, tamē kyā jaī rahyā chō? क्या तुम ऑफिस जा रहे हो?"

Was accurately transcribed as:

"Hello, tame kya jai rahyacho? kya tum office ja rahe ho?"

An average recognition accuracy of 94.2% was achieved across 300 sentences. Recognition accuracy remained stable even with mid-sentence language switching, indicating the robustness of the underlying speech-to-text engine.

Mapping Accuracy

The speech recognition module employs the Google Web Speech API to transcribe real-time multilingual utterances. Recognition accuracy was measured across multiple trials involving code-switched.

Mapping accuracy evaluates how precisely Romanized Hindi and Gujarati words are transformed into their Unicode equivalents using CSV-based dictionaries.

Consider the transcribed output:

“Hello, tamē kyā jaī rahyā chō? kya tum office ja rahe ho?”

After mapping:

"Hello, તમે ક્યાં જઈ રહ્યા છો? क्या तुम ऑफिस जा रहे हो?"

In controlled experiments, mapping accuracy averaged 93.6%, with highest fidelity observed when users adhered to consistent Roman spellings. The mapping mechanism handled most variations via fuzzy matching and partial string comparisons.

Processing Latency

Latency was measured as the time from audio input to final text display. Average end-to-end latency per sentence was 435 milliseconds, including:

- Audio capture: ~120 ms
- Speech recognition: ~200 ms
- Tokenization & mapping: ~115 ms

The system supports real-time transcription with sub-second delays, making it suitable for interactive usage such as lectures or bilingual note-taking.

Language Identification Accuracy

A word-by-word comparison of detected language (before mapping) was performed. The sample sentence used:

"Kal meeting chhe at 11 baje."

Recognized:

“Kal” → Hindi

“meeting” → English

“chhe” → Gujarati

“at” → English

“11” → English

“baje” → Hindi

Table 3 shows the accuracy of language identification.

Table 3: Accuracy of language identification

Language	Precision (%)	Recall (%)	F1-Score (%)
Hindi	94.1	92.8	93.4
Gujarati	93.3	91.7	92.5
English	95.5	96.1	95.8

Sentence Complexity vs Latency Analysis

Table 4: Latency vs Sentence Length

Sentence Length (Words)	Avg. Latency (ms)	Minimum (ms)	Maximum (ms)
5–7	410	372	439
8–12	437	405	468
13–17	455	422	497

To evaluate the scalability and responsiveness of the proposed Multilingual Speech Notepad, we analyzed the relationship between sentence complexity and system latency. Sentence complexity was measured in terms of the total number of words in a single utterance, including all three languages—Hindi, Gujarati, and English. Each category (5–7, 8–12, and 13–17 words) consisted of 20 test sentences recorded under similar environmental conditions. Latency was calculated from the moment speech input began until the final mapped multilingual sentence appeared in the text editor. As shown in the table below, the system maintained an average latency under 500 milliseconds across all sentence groups, demonstrating its real-time performance capabilities even with moderately complex, code-switched input. This proves its applicability in real-world multilingual environments such as classrooms, meetings, and personal note-taking, where mixed-language speech is common.

Table 4 shows the system's average, minimum, and maximum latency across different sentence length categories.

CONCLUSION AND FUTURE SCOPE

This study presented the design, development, and evaluation of a Multilingual Speech Notepad capable of real-time transcription of code-switched speech involving English, Hindi, and Gujarati. The proposed system captures spoken input through the microphone, performs real-time transcription using the Google Web Speech API, and dynamically maps Romanized Hindi and Gujarati words to their corresponding Unicode representations using pre-defined CSV dictionaries. The final output, composed of accurately transcribed multilingual content, is rendered in a Notepad-like text editor, supporting seamless Save, PDF Download, and Share via Gmail/WhatsApp functionalities.

Through systematic experimentation, the system demonstrated high recognition accuracy and mapping precision with minimal latency, even under complex sentence structures and spontaneous code-switching. The architecture also proved to be modular and adaptable, allowing integration with external APIs and support for flexible UI enhancements. These outcomes affirm the system's applicability in real-world multilingual scenarios, including educational environments, digital note-taking, assistive technologies, and documentation in diverse linguistic settings.

Despite its promising performance, the system opens several avenues for further research and enhancement:

Multi-Language Expansion: Incorporate additional Indian languages (e.g., Tamil, Bengali, Marathi) by extending the word-mapping CSV repositories to improve regional inclusivity.

Offline Functionality: Integrate local ASR engines such as Vosk or Whisper to allow speech transcription without internet dependency, enhancing usability in remote or low-bandwidth areas.

Semantic Understanding & Punctuation: Employ NLP techniques for automatic punctuation, sentence segmentation, and better formatting, enhancing the naturalness and readability of output.

Mobile Application Porting: Transform the desktop prototype into an Android/iOS application to facilitate portable, real-time multilingual note-taking across devices.

Voice Command Integration: Enable voice-activated commands like "save note", "share file", or "clear text" to improve accessibility, especially for users with motor impairments.

Real-Time Collaboration: Develop features that allow shared multilingual notes over cloud platforms in real-time for collaborative environments such as classrooms or meetings.

This work lays a foundational framework for multilingual speech-enabled applications in Indian language contexts and offers a significant step toward inclusive, intelligent, and speech-driven human-computer interaction.

REFRENCES

[1] Liu, J., et al. (2024). A computer-assisted interpreting system for multilingual conferences based on ASR. IEEE Access.
[2] Singh, N. K., et al. (2024). MECOS: A bilingual Manipuri–English speech corpus for ASR. Computer Speech & Language.
[3] Huang, K.-P., et al. (2024). Zero resource code-switched speech benchmark. In Proceedings of the IEEE ICASSP.
[4] Madan, C., et al. (2024). CodeFed: Federated speech recognition for low-resource code-switching. ACM Transactions on Asian and Low-Resource Language Information Processing.
[5] Jain, P., & Bhowmick, A. (2024). VITB-HEBiC: A corpus for Indian CS scenarios. Applied Acoustics.
[6] Xi, Y., et al. (2024). Semi-supervised CS-ASR with LLM filtering. In Proceedings of the IEEE SLT Workshop.
[7] Palivela, H., et al. (2025). Hindi-Marathi code-switching ASR. IEEE Access.
[8] Ugan, E. Y., et al. (2025). PIER: A metric for evaluating CS-ASR. arXiv preprint arXiv:2025.
[9] Pham, D., et al. (2025). AdaCS: Adaptive normalization in CS-ASR. arXiv preprint arXiv:2025.
[10] Haboussi, S., et al. (2025). Arabic ASR using neural networks. Umm Al-Qura Journal of Engineering and Architecture.
[11] Singh, N. K., et al. (2025). Synthetic data for Manipuri-English CS-ASR. IEEE Access.
[12] Orosoo, M., et al. (2025). MLP-LSTM ASR for language learning. Alexandria Engineering Journal.
[13] Hamamra, B., et al. (2025). Code-switching and identity. Cogent Arts & Humanities.
[14] Wang, G., et al. (2025). Multilingual transcription for video conferences. Spectrum Research.
[15] Yellamma, P., et al. (2024). Multilingual ASR with Google Cloud API. In Proceedings of ICMCSI.
[16] Lyu, K.-M., et al. (2024). Real-time multilingual ASR with Whisper. PeerJ Computer Science.
[17] Gavino, M. F., & Goldrick, M. (2024). Perception of CS-speech in noise. JASA Express Letters.
[18] Padmane, P., et al. (2022). Multilingual ASR and translation. International Journal of Innovative Engineering and Science.
[19] Jani, M. M., et al. (2023). Multilingual ASR: Challenges and applications. In Proceedings of Springer ICIS.
[20] Patel, H. H., & Sureja, N. M. (2021). UML tool for distributed software. Turkish Journal of Computer and Mathematics Education.
[21] Zhou, J., et al. (2025). CS-Dialogue: Mandarin-English dataset. arXiv preprint arXiv:2025.
[22] Shankar, B. S., et al. (2025). CoSTA: Code-switched speech translation. In Proceedings of COLING.
[23] Yeo, J. H., et al. (2025). Zero-AVSR with LLMs. arXiv preprint arXiv:2025.
[24] Liu, H., et al. (2023). Interactive language bias for CS-ASR. arXiv preprint arXiv:2023.
[25] Dhawan, K., et al. (2023). Unified model with tokenizer for CS-ASR. In Proceedings of CALCS.
[26] Dalmia, S., et al. (2020). Transformer-transducers for CS-ASR. arXiv preprint arXiv:2020.
[27] Kumar, M. G., et al. (2021). Dual script E2E for multilingual ASR. arXiv preprint arXiv:2021.
[28] Aditya, B., et al. (2023). Attention-guided CS-ASR adaptation. arXiv preprint arXiv:2023.
[29] Zhou, J., et al. (2024). Zero-shot CS-ASR with kNN-CTC. arXiv preprint arXiv:2024.
[30] Kulkarni, S. S., et al. (2020). End-to-end CS language models for ASR. arXiv preprint arXiv:2020.
[31] Aggarwal, D., et al. (2024). Hindi-English code switching with LSTM. International Journal of Engineering Science and Research Technology.
[32] Chandrasekaran, V., et al. (2024). On the need for multilingual ASR systems in education. IEEE EdSoc.
[33] Ismaiel, W., et al. (2024). Deep learning for Arabic speech emotion recognition. International Journal of Computer Science and Network Security.
[34] Muthukumar, K., et al. (2025). Enhancing ASR accuracy with acoustic modeling. In Proceedings of Springer LNCS.

[35] Rana, M., et al. (2024). Real-time speech-to-text on low-end devices. International Journal of Computer Applications.
[36] Chanu, Y. J., et al. (2025). Multi-accent Hindi-English ASR. Computers & Electrical Engineering.
[37] Goyal, M., et al. (2024). Cross-lingual ASR using joint embeddings. In Proceedings of the IEEE ICASSP.
[38] Shastri, A., et al. (2025). Building robust Hindi ASR models with Whisper. AI Open.
[39] Pidugu, M., et al. (2025). Emotion-aware ASR using hybrid neural networks. Procedia Computer Science.
[40] Khamaru, S., et al. (2025). Language-specific feature adaptation in ASR. IEEE Access.
[41] Chatterjee, A., et al. (2024). Real-time multilingual voice assistants. International Journal of Advanced Computer Intelligence.