Real-Time Language Translation Using AI

A PROJECT REPORT

Submitted by

Mr. MOHAMMED NIHAL A S - 20211CSE0367
Mr. DARSHAN A R – 20211CSE0352
Mr. MAHESH G – 20211CSE0357
Mr. SANDESH W D – 20211CSE0364
Mr. ROHAN S HANDRAL – 20211CSE0389

Under the guidance of

Dr. VIJAYAKUMAR ADAICKALAM

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

at

PRESIDENCY UNIVERSITY, BENGALURU(560064)

DECEMBER 2024

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

This is to certify that the Project report **"Real-Time Language Translation Using WhisperAI"** being submitted by "Mohammed Nihal A S, Darshan A R, Mahesh G, Sandesh W D, Rohan S Handral" bearing Roll Number(s) "20211CSE0367, 20211CSE0352, 20211CSE0357, 20211CSE0364, 20211CSE0389" in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a bonafide work carried out under my supervision.

Dr VIJAYAKUMAR ADAICKALAM
Professor, School of CSE&IS
Presidency University,
Bangalore-560064

Dr. MOHAMMED ASIF T
HOD, School of CSE&IS
Presidency University,
Bangalore-560064

Dr. L. SHAKKEERA
Associate Dean
School of CSE
Presidency University

Dr. MYDHILI NAIR
Associate Dean
School of CSE
Presidency University

Dr. SAMEERUDDIN KHAN
Pro-Vc School of Engineering
Dean -School of CSE&IS
Presidency University

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

DECLARATION

We hereby declare that the work which is being presented in the project report entitled **Real-Time Language Translation Using WhisperAI** in partial fulfillment for the award of Degree of **Bachelor of Technology** in **Computer Science and Engineering** is a record of our own investigations carried under the guidance of **Dr. VIJAYAKUMAR ADAICKALAM, Professor, School of Computer Science Engineering & Information Science, Presidency University, Bengaluru.**

We have not submitted the matter presented in this report anywhere for the award of any other Degree.

MOHAMMED NIHAL A S – 20211CSE0367
DARSHAN A R – 2021CSE0352
MAHESH G – 20211CSE0357
SANDESH W D – 20211CSE0364
ROHAN S HANDRAL – 20211CSE0389

ABSTRACT

Real-time transcription and translation systems play a pivotal role in reducing language barriers and enhancing global communication and develop a user-friendly application leveraging advanced deep learning technology for real-time transcription and translation. The envisioned application is designed for use in professional environments (such as remote work) as well as leisure activities like watching videos. At present, there is likely no other application that applies automatic speech recognition in this manner. The closest comparable tools, such as Google Translate, are not intended for real-time use on computers; they require users to provide input and only display results after processing is complete. To address this gap, a desktop application was developed that integrates OpenAI's Whisper model for transcription and Google translation for translation within a single, user-friendly graphical interface. The project employed an iterative and incremental approach for both the GUI design and software development phases. In the end, the focus will be on developing a successful project that meets the objectives of real-time transcription and translation, providing a seamless experience for users that can be applied to digital meetings or online videos.

ACKNOWLEDGEMENT

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC, School of Engineering and Dean, School of Computer Science Engineering & Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Deans **Dr. Shakkeera L and Dr. Mydhili Nair,** School of Computer Science Engineering & Information Science, Presidency University, and Dr. "MOHAMMED ASIF T", Head of the Department, School of Computer Science Engineering & Information Science, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Dr Vijayakumar Adaickalam, Professor** and Reviewer **Dr Jeyanthi Kameleswaran**, School of Computer Science Engineering & Information Science, Presidency University for her inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the project work.

We would like to convey our gratitude and heartfelt thanks to the PIP2001 Capstone Project Coordinators **Dr. Sampath A K, Dr. Abdul Khadar A and Mr. Md Zia Ur Rahman,** department Project Coordinators and Git hub coordinator **Mr. Muthuraj.**

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

Mohammed Nihal A S

Darshan A R

Mahesh G

Sandesh W D

Rohan S Handral

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER-1
INTRODUCTION

In recent years, deep learning technology has seen rapid growth and innovation. These neural networks are designed to work like the human brain, not by following specific instructions but by learning from large amounts of data. As they process these data and receive feedback through labeled examples, they gradually improve their performance.

Deep learning is now used in many areas, such as self-driving cars and advanced chatbots like ChatGPT, which use a type of neural network called a Transformer. Introduced in 2017, the Transformer model has become essential for tasks involving image and speech recognition.

1.1 Background

Automatic speech recognition (ASR) has come a long way since the 1950s, when it could only perform simple tasks. Over the years, ASR systems have improved significantly, becoming more accurate and capable. The Transformer model has been a big breakthrough, allowing better context understanding and improved transcription and translation.

One of the most advanced ASR systems is OpenAI's Whisper. This model provides nearly human-level transcription for English and supports 99 other languages with different levels of accuracy. It was trained on 680,000 hours of labeled audio data. However, Whisper typically processes audio as a whole and only produces results after the full file has been analyzed.

1.2 Problem Statement

Even though Whisper is powerful, it does not support real-time transcription and translation for ongoing conversations. Also, its command-line interface can be difficult for non-technical users to navigate. Most current ASR solutions do not provide real-time, user-friendly, offline-capable applications that can be used easily on a computer.

1.3 Purpose

The purpose of this research is to adapt OpenAI's Whisper model for real-time transcription and translation while keeping its high accuracy. The goal is to see if the model can be used effectively for real-time applications.

1.4 Goals

This project has three main goals:

1.4.1 Accuracy: To make sure if the application maintains good transcription and translation accuracy, using the word error rate (WER) as a metric.

1.4.2 Speed: Ensuring the app works in real-time by measuring latency (the time taken from input to output) and throughput (words processed per minute).

1.4.3 User-Friendly Design: Creating a simple and accessible user interface so that people with different technical skills can use the application easily.

CHAPTER-2

LITERATURE SURVEY

2.1 Technologies used in the paper:

2.1.1 V. R. and I. A. Funcke (2023) introduce aiLangu, a system that allows speech transcription and translation in real time, thereby lowering language barriers. The writers stress the value of accessibility as well as the difficulties in creating reliable, latency-effective solutions that work in a variety of linguistic and auditory contexts. This study emphasizes how transcription and translation can be combined to facilitate smooth communication in multilingual environments. For real-time applications, the suggested framework acts as a fundamental manual for combining Whisper AI with translation programs like Google Translate. [1]

2.1.2 Y. Peng et al. (2023) investigate Whisper-style model training techniques with open-source toolkits and publicly accessible data. They utilized transformer-based architectures to replicate Whisper's functionality and leveraged frameworks such as PyTorch and Hugging Face's Transformers library for model development and experimentation. Their research demonstrates how Whisper AI's end-to-end architecture and strong multilingual capabilities make it an excellent choice for transcribing tasks. The authors address the difficulties in reproducing private models and suggest methods for using public datasets to duplicate Whisper's training paradigm. Whisper AI is a good option for real-time voice translation projects because of the insightful information this study offers about its architecture [2].

2.1.3 D. Wang et al. (2019) provided a thorough description of automatic speech recognition (ASR) systems from start to finish. They highlight the use of transformer-based models, such as the Transformer and Conformer architectures, which enhance sequence-to-sequence learning. The paper also discusses frameworks like TensorFlow and Kaldi that have been widely adopted in developing ASR systems. They talk about the switch from conventional pipeline-based ASR systems to end-to-end strategies that increase efficiency and simplify the design. The study explores a number of issues, including as scalability across languages, robustness in noisy situations, and the significance of extensive pretraining. Their research is highly relevant to the use of Whisper AI, an end-to-end ASR model, in real-time applications where high accuracy and

resilience are required [3].

2.1.4 D. Macháček et al. (2024) look into what needs to be changed to make Whisper AI a real-time transcribing system. The study highlights the use of transformer-based architectures to optimize the model's latency and performance. Frameworks like PyTorch were employed to implement and evaluate the proposed enhancements, ensuring the system's scalability and real-time efficiency. In order to modify Whisper's architecture to satisfy the requirements of real-time applications, they address latency issues and provide optimization techniques. The study's emphasis on speeding up response times without sacrificing transcription quality offers a model for incorporating Whisper AI into speech translation systems, where a low latency is essential [4].

2.1.5 A popular global translation service, Google Translate continues to set the standard for accessibility and translation accuracy. It employs neural machine translation (NMT) techniques, leveraging transformer-based architectures such as the Transformer model to process and translate text efficiently. The platform utilizes TensorFlow as a primary framework for training and deploying its models The platform is an essential part of voice translation systems because of its frequent updates and support for more than 100 languages. Real-time speech-to-speech translation systems can be developed thanks to its API capabilities, which provide smooth integration into transcription pipelines [5].

2.1.6 An open-source translation program called LibreTranslate has more customization options than Google Translate but still provides comparable functionality. It uses neural machine translation (NMT) techniques and leverages frameworks such as OpenNMT and PyTorch for its implementation, ensuring flexibility and efficiency in translation tasks. Without depending on proprietary APIs, it gives developers the freedom to create translation systems. By ensuring that the system can operate independently of commercial platforms, LibreTranslate is incorporated into voice translation initiatives, offering a financially viable alternative for research and development [6].

2.1.7 B. Raj and R. Olivier (2022) talk about how resilient Whisper AI is to suspicious cases. Their study reveals weaknesses in Whisper's transcriptional capacities under precisely calibrated disturbances. This study emphasizes how crucial it is to strengthen transcription models' resilience, especially in real-time speech translation systems where

adversarial noise may affect efficiency. In order to guarantee the dependability and security of systems that use Whisper AI, these issues must be resolved [7].

2.1.8 M. Aiken (2021) offers a comprehensive assessment of Google Translate's accuracy in a variety of languages and fields. The report identifies enduring difficulties in less popular or domain-specific translations while highlighting advancements in translation quality, especially in widely used languages. The importance of extensive training data in enhancing translation accuracy and dependability is shown by Aiken's work. This work highlights the significance of integrating transcription accuracy with strong translation skills and provides insightful information about Google Translate's possible integration with Whisper AI for multilingual voice translation systems [8].

2.2 Challenges faced in the paper :

2.2.1 Latency and Real-Time Processing :

Real-time transcription and translation are still difficult because current models like OpenAI's Whisper were made for processing whole audio files at once, not for streaming live audio. There's always some delay in these systems because processing sound data takes time.

2.2.2 Usability Barriers :

Many systems, like Whisper, don't have easy-to-use interfaces and depend on command-line operations, which makes them hard for non-technical users to access.

2.2.3 Model Limitations :

Performance varies a lot between languages. Widely spoken languages like English and Spanish are more accurate compared to less common ones like Slovenian or complex ones like Arabic. Whisper mainly translates into English and has limited support for translating back and forth between languages.

2.2.4 Technical Constraints :

Whisper relies on NVIDIA GPUs (using CUDA), which makes it less accessible for

users with AMD hardware. For real-time use, the audio needs to be split into smaller parts, adding complexity to the setup.

2.3 Advancements found in the Existing System :

2.3.1 Integration of Existing Tools :

Using OpenAI's Whisper for transcription and Argos Translate for translation offers a flexible way to do real-time translation, even without an internet connection.

2.3.2 Improved Speech Models :

Whisper is a big step forward in automatic speech recognition (ASR) technology. It uses a huge amount of audio data (680,000 hours) to train, which helps it achieve almost human-like accuracy in certain languages.

CHAPTER-3

RESEARCH GAPS OF EXISTING METHODS

The research gaps for this study would focus on the limitations of existing methods in real-time automatic transcription and translation.

3.1 Lack of Real-Time Processing:

Most current ASR models, including Whisper, are designed to process entire audio files, resulting in delayed transcription and translation. Existing tools like Google Translate do not support seamless real-time processing for live audio.

3.2 Dependence on Internet Connectivity:

Many high-performing ASR and translation tools require a constant internet connection, which can be a barrier for users in areas with poor connectivity. Integrating models that work efficiently offline could provide more flexibility.

3.3 Limited Language Coverage in Real-Time:

While some models support multiple languages, real-time translation between less commonly used languages, especially offline, is still limited and requires more research and development.

3.4 High Latency Issues:

Existing real-time ASR implementations often face challenges with latency, which affects the practical usability in scenarios such as live meetings or video consumption. Reducing latency while maintaining accuracy remains a challenge.

3.5 Balancing Speed and Accuracy:

Ensuring that real-time transcription maintains a low word error rate (WER) without sacrificing speed is a significant gap in current solutions. This balance is essential for effective real-time applications.

CHAPTER-4

PROPOSED SYSTEM

System Architecture:

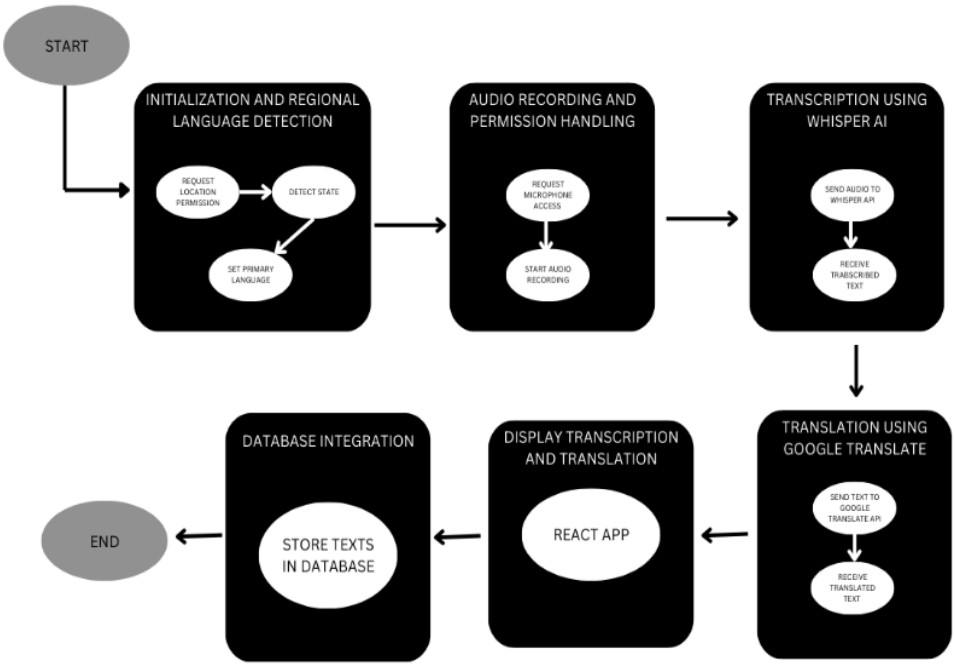


Fig 4.1 System Architecture

4.1 Using Whisper AI for Main ASR:

Integrate Whisper AI to frontend to convert speech to text. It supports multiple languages and excels in real-time transcriptions. Whisper can handle general language translation tasks seamlessly.

4.2 Data Collection and Preprocessing:

Multilingual Audio Dataset: Collect a large and diverse set of audio recordings in different languages, accents, and dialects to ensure the model can handle various inputs.

Transcription and Translation: Convert the audio data into text and translate it into the target languages using existing ASR (Automatic Speech Recognition) and translation models.

4.3 Real-Time Processing Pipeline:

Speech Recognition: Implement a system to convert spoken language into text in real-time, capable of handling different accents and dialects.

Translation: Use a neural machine translation (NMT) model to translate the recognized text into the target language, optimized for low latency.

Text-to-Speech (TTS): Convert the translated text back into speech, ensuring it sounds natural.

4.4 Optimization and Latency Reduction:

Streaming and Chunking: Process audio in small segments to reduce delays and deliver translations quickly.

Parallel Processing: Run ASR, translation, and TTS tasks simultaneously to minimize delays.

4.5 Integration and Deployment:

User Interface: Develop a user-friendly interface for the translation system, which could be a mobile app, desktop application, or web-based platform.

4.6 API Integration:

Connect the system with APIs for smooth communication between different components.

4.7 Testing and Validation:

Test the system extensively to ensure it works well in various real-world scenarios, including different languages and accents.

CHAPTER-5

OBJECTIVES

5.1 Automatically detect user location on startup to set the regional input language:

On startup, the application will request the user's location (e.g., through geolocation APIs). Based on the user's region, the app will automatically assign a corresponding regional language as the default input language. This improves usability and eliminates the need for manual setup.

5.2 Accept input via voice:

The application will allow users to input their desired content through either voice recording or typing. This flexibility ensures accessibility for users with different preferences or needs.

5.3 Reduce ambient noise for clearer input:

To improve the quality of voice input, the application will employ noise-reduction techniques or algorithms. This ensures that the speech signal is as clear as possible, leading to more accurate transcription and translation results.

5.4 Utilize ASR (Automatic Speech Recognition) models for transcription:

ASR models like Whisper AI library will be used to convert the user's voice input into text accurately. These models will play a central role in processing and understanding spoken language.

5.5 Generate accurate output translations:

The app will take the transcribed input and provide an appropriate translation in the target language, ensuring that the output meets the user's expectations.

5.6 Save translation history for quick access:

The application will maintain a record of past translations. This feature allows users to revisit previous translations without needing to re-enter input, improving convenience and efficiency.

5.7 Optionally implement user authentication:

If authentication is enabled, users can log in to access their saved history across

devices, enhancing the app's personalization and security. However, this feature is optional and can be implemented based on user requirements.

CHAPTER-6

SYSTEM DESIGN & IMPLEMENTATION

The proposed system is designed to provide a efficient real time translation, focusing on integrating advanced speech recognition and translation models into a user -friendly application.

Figure 6.1 : System function flowchart

6.1 Architectural Design (fig-4.1):

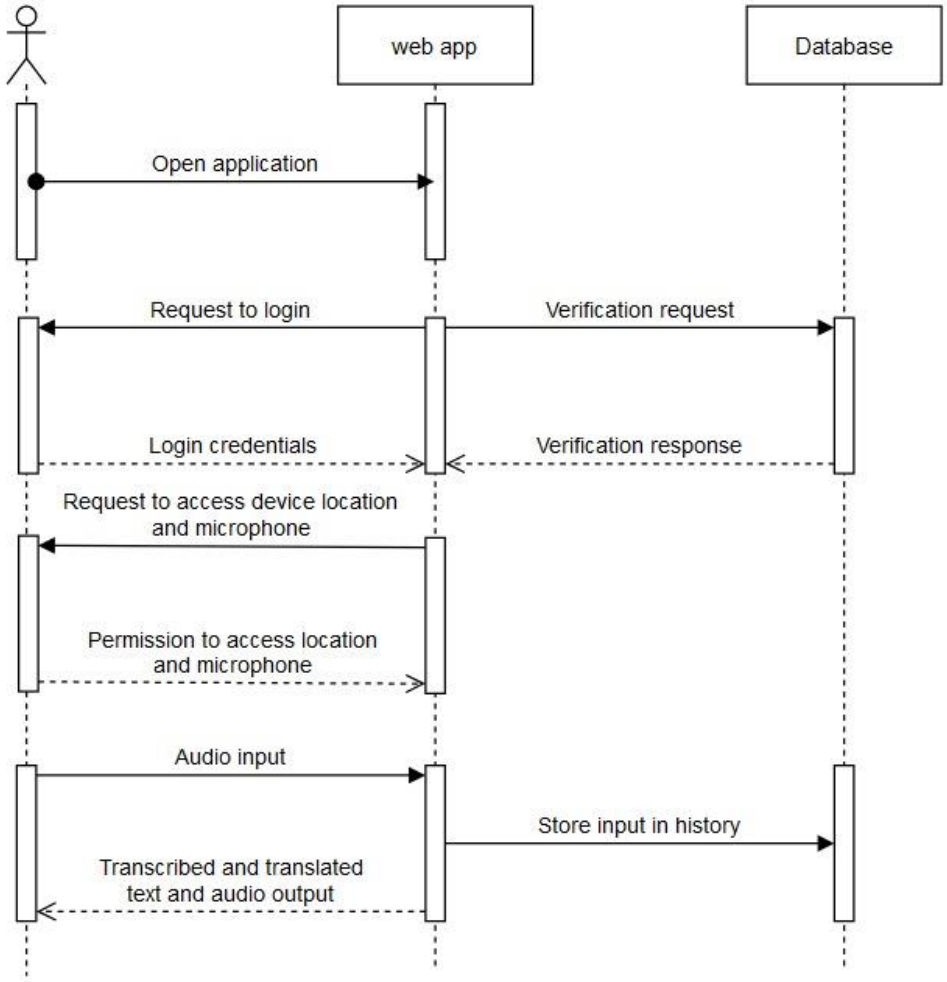


Figure 6.2 : Sequence diagram

The system follows a three-tier architecture :

Frontend Layer : A user-friendly interface built using ReactJS bridges the gap between complex back-end processes and user interaction. Helps the tourists who doesn't know the regional languages or a person who has newly shifted to the state.

Backend Layer : A robust API service developed using Node.js and Python Flask server which handles the translation and transcription.

Database Layer : MongoDB serves as the database to store the audio files and the MongoDB structures as a JSON.

6.2 Database Design :

MongoDB prompt collection stores all the translation query from the user making it accessible to the user quickly.

6.3 Technology Stack :

Frontend : ReactJS for a dynamic and responsive user experience.

Backend : Node.js and Python Flask for translation and transcription.

Database : MongoDB for secure and efficient data storage.

AI Integration : Whisper AI library provides nearly human-level transcription and It was trained on 680,000 hours of labeled audio data.

6.4 Implementation :

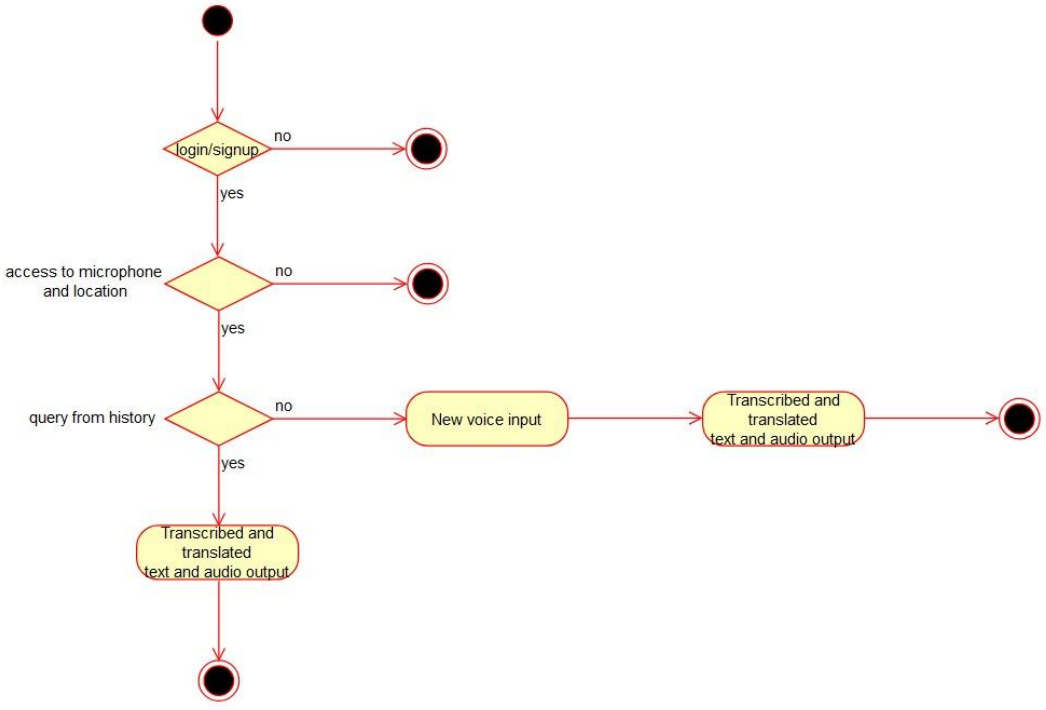


Figure 6.3 : Activity Diagram

6.4.1 Frontend Development :

Developed with ReactJS for a dynamic and responsive user experience.

User Interface requires geo location as the preffered language for the output.

6.4.2 Backend Development :

RESTful APIs are created for interaction with the database, ensuring secure and efficient data transfer.

APIs support updating the prompt, uploading the audio files, providing the preview of the audio file and transferring the translation back to the frontend.

6.4.3 AI Integration :

Whisper AI library provides nearly human-level transcription and there were six model sizes out of which we chose the medium model .It was trained on 680,000 hours of labeled audio data and also it enhances the user interaction and provides the expected translation and the transcription.

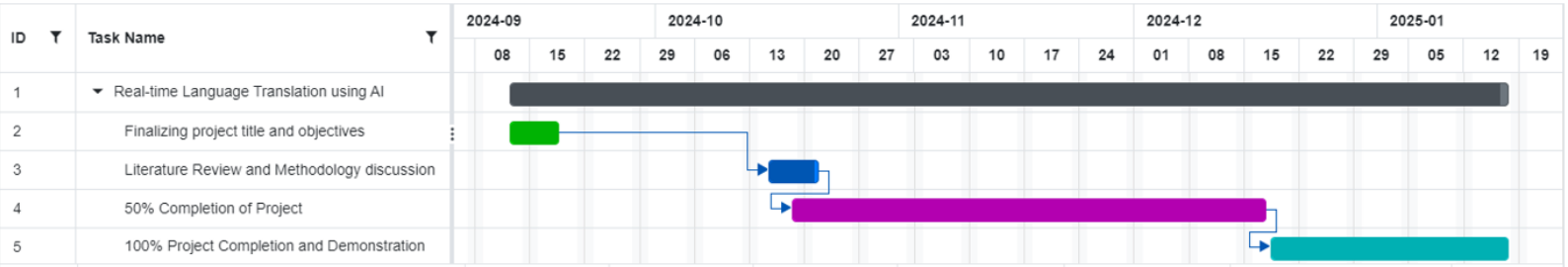
6.4.4 Testing and Debugging :

Unit and integration testing ensure the reliability of individual components and overall system performance. Regular debugging addresses potential issues and ensures system stability.

CHAPTER-7

TIMELINE FOR EXECUTION OF PROJECT

(GANTT CHART)



ID	Task Name	Start	End	Duration	Progress %	Dependency	Resources	Color
1	Real-time Language Translation using AI	2024-09-12	2025-01-17	92 days	97			
2	Finalizing project title and objectives	2024-09-12	2024-09-18	5 days	100			
3	Literature Review and Methodology discussion	2024-10-15	2024-10-21	5 days	90	2FS+18 days		
4	50% Completion of Project	2024-12-17	2024-12-20	4 days	100	3FS+40 days		
5	100% Project Completion and Demonstration	2025-01-10	2025-01-17	6 days	100	4FS+14 days		

Figure 7.1 : Gantt Chart

CHAPTER-8

OUTCOMES

8.1 Accurate Translations:

The tool should effectively translate text between multiple languages with high accuracy, capturing both literal meaning and context.

8.2 User-Friendly Interface:

A simple and intuitive user interface, where users can easily input text and view the translated output.

8.3 Real-Time Processing:

Quick and responsive translation, ideally offering near-instant results for smaller text inputs.

8.4 Support for Multiple Languages:

A broad range of languages, including both widely spoken and less common languages, based on user needs.

8.5 Scalability and Performance:

The tool should be able to handle varying loads, from single users to many concurrent users, without performance issues.

8.6 Integration Capabilities:

Potential for integration with other software or platforms (e.g., websites, mobile apps, or chatbots).

8.7 Customizable Translation Options:

Options to customize translations for specific industries or contexts (e.g., legal, medical, technical).

8.8 Security and Privacy:

Protection of user data and compliance with data privacy regulations.

CHAPTER-9

RESULTS AND DISCUSSIONS

The proposed model provides insights into the into the performance and implications of the real-time transcription and translation system. The results demonstrate significant improvements in key areas:

9.1 Performance Metrics :

Accurate translation:

The system achieved reasonable accuracy for widely spoken languages such as English.

Latency :

The latency from input to transcription output was within acceptable limits.

9.2 Comparison Factors :

Comparing OpenAI Whisper with other ASR models(such as Wav2Vec , LibriSpeech-trained ASRs)

Whisper's accuracy, latency, throughput, and speed, especially in real-world multi-lingual applications, makes it a superior choice compared to other ASR models, particularly when privacy and offline usage are critical. However, its hardware demands and language-specific performance gaps remain areas for improvement.

9.3 Enhanced resources :

Seamless database response – It has a faster callback response from the database.

9.4 User Interaction and Accessibility :

The GUI significantly enchances the usability , opening up the application to a broader audience. Making it accessible to the people who cannot understand regional languages.

9.5 Social Impact:

The social impact of the project revolves around its potential to significantly reduce language barriers and improve accessibility and communication in various societal

contexts and also facilitates effective communication among teams in multinational companies, enabling seamless interaction between speakers of different languages. Individuals with limited language skills or disabilities can access new job opportunities where communication is a critical requirement

CHAPTER-10

CONCLUSION AND FUTURE ENHANCEMENT

Creating real-time transcription and translation applications is important for overcoming language barriers and improving global communication. This research looked at how OpenAI's Whisper model could be adapted to build a user-friendly desktop app for real-time transcription and translation. The review showed major improvements in ASR technology, such as the development of the Transformer model and the strong performance of the Whisper model. However, current solutions have issues like not being able to process in real-time, complicated user interfaces, and needing an internet connection. This project aims to fill these gaps by adapting the Whisper model for real-time transcription that is both accurate and quick. The main goal is to develop an app that is easy to use and lets people take advantage of advanced technology without needing technical skills.

10.1 Future Directions :

Future work will focus on improving performance, such as lowering word error rate (WER) and latency, and ensuring the app can work offline to make it more flexible. If successful, this project could greatly improve real-time multilingual communication for work and personal use, helping people connect more easily around the world.

10.1.1 Expanding Language Support :

Efforts are being made to train models on more varied datasets to improve performance for less common languages.

10.1.2 Optimizing for Resource-Constrained Devices :

Reducing the need for high-end hardware like CUDA GPUs to make the solution easier to use for more people.

10.1.3 Enhanced Real-Time Capabilities :

Further improving speed and efficiency to provide smooth real-time interaction, possibly using advanced streaming methods.

10.1.4 Mobile and Cross-Platform Usability :

Extending the application to mobile platforms, ensuring accessibility and usability across devices.

REFERENCES

[1] V. R. and I. A. Funcke, "aiLangu – Real time Transcription and Translation to Reduce Language Barriers," KTH Royal Institute of Technology, 2023.

[2] Y. Peng et al., "Reproducing Whisper Style Training Using An Open-Source Toolkit And Publicly Available Data," 2023, doi: 10.1109/ASRU57964.2023.10389676.

[3] D. Wang, X. Wang, and S. Lv, "An overview of end-to-end automatic speech recognition," Symmetry. 2019, doi: 10.3390/sym11081018.

[4] D. Macháček, R. Dabre, and O. Bojar, "Turning Whisper into Real-Time Transcription System," 2024, doi: 10.18653/v1/2023.ijcnlp-demo.3.

[5] [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.translate&hl=en&gl=US.

[6] [Online]. Available: https://github.com/LibreTranslate/LibreTranslate.

[7] R. Olivier and B. Raj, "There is more than one kind of robustness: Fooling whisper with adversarial examples," arXiv:2210.17316, 2022.

[8] M. Aiken, "An Updated Evaluation of Google Translate Accuracy," School of Business Administration, University of Mississippi, 2021.

APPENDIX-A

PSUEDOCODE

Recorder.js

```
import { useAuthContext  } from '../hooks/useAuthContext';
import React, { useState, useRef, useEffect } from "react";
import Sidebar from "./Sidebar";
import ToggleButton from "./langSelector";
import LanguageSelector from "./LanguageSelector";
import './Recorder.css'

const Recorder = () => {
  const [isRecording, setIsRecording] = useState(false);
  const [audioBlob, setAudioBlob] = useState(null);
  const [uploadStatus, setUploadStatus] = useState("");
  const { user } = useAuthContext();
  const [originalTranscript, setOriginalTranscript] = useState("");
  const [translatedTranscript, setTranslatedTranscript] = useState("");
  const [isSidebarVisible, setIsSidebarVisible] = useState(false);
  const [prompts, setPrompts] = useState([]); // Store fetched prompts
  const [userLanguage, setUserLanguage] = useState("en"); // Source language
  const [binodLanguage, setBinodLanguage] = useState("en"); // Target language
  const [isToggled, setIsToggled] = useState(false);
  const [error, setError] = useState(null);

  const mediaRecorder = useRef(null);
  const audioChunks = useRef([]);

  // Language change handlers
  const handleUserLanguageChange = (language) => setUserLanguage(language);
  const handleBinodLanguageChange = (language) => setBinodLanguage(language);

  const handleToggle = () => setIsToggled(!isToggled);
  const reverseLanguages = () => {
```

```
  const temp = userLanguage;
  setUserLanguage(binodLanguage);
  setBinodLanguage(temp);
};

useEffect(() => {
  if (originalTranscript && translatedTranscript) {
    savePromptToDatabase(originalTranscript, translatedTranscript);
  }
}, );

// useEffect(() => {
//   fetchPrompts(); // Fetch history on mount
// }, []);

const toggleSidebar = () => setIsSidebarVisible(!isSidebarVisible);

const startRecording = async () => {
  try {
    const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
    mediaRecorder.current = new MediaRecorder(stream);

    mediaRecorder.current.ondataavailable = (event) => {
      if (event.data.size > 0) {
        audioChunks.current.push(event.data);
      }
    };

    mediaRecorder.current.onstop = () => {
      const blob = new Blob(audioChunks.current, { type: "audio/wav" });
      setAudioBlob(blob);
      audioChunks.current = [];
    };
```

```
    mediaRecorder.current.start();
    setIsRecording(true);
  } catch (err) {
    console.error("Error accessing audio devices: ", err);
  }
};
const fetchPrompts = async () => {
  if (user) {
    try {
      const response = await fetch("http://localhost:4000/api/prompts", {
        headers: {
          Authorization: `Bearer ${user.token}`,
        },
      });
      const data = await response.json();
      if (response.ok) {
        setPrompts(data);
      }
    } catch (error) {
      console.error("Failed to fetch prompts:", error);
    }
  }
};

const stopRecording = () => {
  if (mediaRecorder.current && isRecording) {
    mediaRecorder.current.stop();
    setIsRecording(false);
  }
};

const sendForTranscriptionAndTranslation = async () => {
  if (!audioBlob) {
    alert("No audio recorded. Please record audio first.");
```

```
    return;
  }

  const formData = new FormData();
  formData.append("file", audioBlob, "recording.wav");
  formData.append("model", "whisper-1");

  try {
    setUploadStatus("Transcribing...");
                                const transcriptionResponse = await
fetch("https://api.openai.com/v1/audio/transcriptions", {
      method: "POST",
      headers: {
                                                Authorization: `Bearer sk-
f4WoZTwX68sKrCD0_sJS6j5c0qKYKCu72TDIspjWDST3BlbkFJxf_0kASq1kiv-
pEfnpn3diL_5dYtkltPjFdreKxlEA`, // Replace with your OpenAI API key
      },
      body: formData,
    });

    if (transcriptionResponse.ok) {
      const transcriptionData = await transcriptionResponse.json();
      const originalText = transcriptionData.text;
      setOriginalTranscript(originalText);

      setUploadStatus("Translating...");
     const translationResponse = await fetch("http://localhost:4000/api/translations/translate",
{
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          text: originalText,
```

```
          userLanguage: userLanguage,
          binodLanguage: binodLanguage,
        }),
      });

      if (translationResponse.ok) {
        const translationData = await translationResponse.json();
        setTranslatedTranscript(translationData.translation.translation);
        setUploadStatus("Transcription and translation completed.");
      } else {
        const errorData = await translationResponse.json();
        setUploadStatus(`Translation Error: ${errorData.error}`);
      }
    } else {
      const errorData = await transcriptionResponse.json();
      setUploadStatus(`Transcription Error: ${errorData.error.message}`);
    }
  } catch (error) {
    console.error("Error processing audio:", error);
    setUploadStatus("Failed to process audio.");
  }
};
  //saving the prompt to the database
const savePromptToDatabase = async (originalTranscript, translatedTranscript) => {
    if(!user) {
      setError('You are not loggen in')
      return
    }

    try {
      const response = await fetch("http://localhost:4000/api/prompts/prompt", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
```

```
          'Authorization': `Bearer ${user.token}`
        },
        body: JSON.stringify({ originalTranscript, translatedTranscript }),
      });

      if (response.ok) {
        console.log("Saved prompt to database.");
        fetchPrompts(); // Fetch prompts after saving
      } else {
        const errorData = await response.json();
        console.error("Failed to save prompt:", errorData.error);
      }
    } catch (error) {
      console.error("Error connecting to the database:", error);
    }

};

const handleDeletePrompt = (id) => {
  // Remove the deleted prompt from the state
  setPrompts((prevPrompts) => prevPrompts.filter((prompt) => prompt._id !== id));
};

const playTranslatedSpeech = async () => {
  if (!translatedTranscript) {
    alert("No translated text available. Please transcribe and translate first.");
    return;
  }

  try {
    const response = await fetch("http://localhost:4000/api/tts", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
```

```
      },
      body: JSON.stringify({
        text: translatedTranscript,
        language: "hi", // Target language for TTS
      }),
    });

    if (response.ok) {
      const data = await response.json();
      const audio = new Audio(data.audioUrl);
      audio.play();
    } else {
      const errorData = await response.json();
      alert(`TTS Error: ${errorData.error}`);
    }
  } catch (error) {
    console.error("Error playing translated speech:", error);
  }
};


 return (
  <div className="recorder" style={{ paddingTop: "10px" }}>
    <button className="sidebar-toggle-button" onClick={toggleSidebar}>
    {isSidebarVisible ? "X" : "History"}
    </button>
                        {isSidebarVisible && <Sidebar prompts={prompts}
onDeletePrompt={handleDeletePrompt} />}
    <h1>Audio Recorder</h1>
    <LanguageSelector
      onUserLanguageChange={handleUserLanguageChange}
      onBinodLanguageChange={handleBinodLanguageChange}
    />
    <button onClick={isRecording ? stopRecording : startRecording}>
```

```
      {isRecording ? "Stop Recording" : "Start Recording"}
    </button>
    <button onClick={sendForTranscriptionAndTranslation} disabled={!audioBlob}>
      Transcribe and Translate
    </button>
    <button onClick={playTranslatedSpeech} disabled={!translatedTranscript}>
      Play Translated Speech
    </button>
    <p>{uploadStatus}</p>
    <p><b>Original Transcript: </b>{originalTranscript}</p>
    <p><b>Translation: </b>{translatedTranscript}</p>
                <ToggleButton isToggled={isToggled} handleToggle={handleToggle}
reverseLanguages={reverseLanguages} />

   </div>
  );
};

export default Recorder;
```

Server.js

```
require('dotenv').config()
// import cors from "cors";

const express = require('express')
const cors = require('cors')
const app = express()
const mongoose = require('mongoose')
const PORT = process.env.PORT;
const promptRouter = require('./routes/promptRouter')
const userRoutes = require('./routes/userRouter')
const translateRoutes = require('./routes/translationRouter')
const gTTS = require("gtts");
```

```
const path=require('path');

app.use(cors());
app.use(express.json());
// app.use(cors())

app.use((req, res, next) => {
   console.log(req.path, req.method)
   next()
})

app.use('/api/prompts', promptRouter);
app.use('/api/user', userRoutes);
app.use('/api/translations', translateRoutes);

// Text-to-Speech Route
app.post("/api/tts", async (req, res) => {
   const { text, language } = req.body;

   if (!text || !language) {
     return res.status(400).json({ error: "Text and language are required" });
   }

   try {
     const tts = new gTTS(text, language);
     const filename = `output_${Date.now()}.mp3`;
     const filepath = path.join(__dirname, "audio", filename);

     // Save the audio file
     tts.save(filepath, (err) => {
       if (err) {
         console.error("Error saving audio file:", err);
         return res.status(500).json({ error: "Failed to generate audio" });
       }
```

```
      res.status(200).json({ audioUrl: `http://localhost:${PORT}/audio/${filename}`});
    });
  } catch (error) {
    console.error("Error generating TTS:", error);
    res.status(500).json({ error: "Error processing TTS request" });
  }
});

// Serve Audio Files
app.use("/audio", express.static(path.join(__dirname, "audio")));
mongoose.connect(process.env.MONGO_URL)
  .then(() => {
    app.listen(PORT, () => {
      console.log(`Connected to the database, Listening at port ${PORT}`)
    })
  })
  .catch((error) => {
    console.log(error);
  })
```

Geo.js

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { ToastContainer, toast } from 'react-toastify';  // Importing ToastContainer and toast from react-toastify
import 'react-toastify/dist/ReactToastify.css';

// Map of places to native languages (you can extend this list)
const placeLanguageMap = {
  "Karnataka": "Kannada",
  "Tamil Nadu": "Tamil",
```

```
    "Telangana": "Telugu",
    "Kerala": "Malayalam",
    "Maharashtra": "Marathi",
    "Goa": "Konkani",
    "Andhra Pradesh": "Telugu",
    "Rajasthan": "Hindi",
    "Haryana": "Hindi",
    "Himachal Pradesh": "Hindi",
    "Uttarakhand": "Hindi",
    "Chhattisgarh": "Hindi",
    "Uttar Pradesh": "Hindi",
    "Bihar": "Hindi",
    "Jharkhand": "Hindi",
    "Madhya Pradesh": "Hindi",
    "Sikkim": "Nepali",
    "Punjab": "Punjabi",
    "West Bengal": "Bengali",
    "Tripura": "Bengali",
    "Gujarat": "Gujarati"
};

const Location = () => {


    const [stateName, setStateName] = useState('');
    const [nativeLanguage, setNativeLanguage] = useState('');
    const [error, setError] = useState('');
    const [toastShown, setToastShown] = useState(false); // Track if toast has been shown



    useEffect(() => {
        const getLocation = () => {
            if (navigator.geolocation) {
```

```
            navigator.geolocation.getCurrentPosition(
              async (position) => {
                const { latitude, longitude } = position.coords;
                await getStateName(latitude, longitude);
              },
              (err) => {
                setError("Error fetching the location: " + err.message);
              }
            );
          } else {
            setError("Geolocation is not supported by this browser.");
          }
        };

        const getStateName = async (lat, lon) => {
          try {
            const response = await axios.get(
              `https://nominatim.openstreetmap.org/reverse?format=json&lat=${lat}&lon=${
lon}`
            );
            const state = response.data.address.state || "State not found";
            setStateName(state);

            // Find the native language based on the state/country
            const language = placeLanguageMap[state] || "Language not found";
            setNativeLanguage(language);

            // Show toast only once when the state and language are fetched
            if (!toastShown) {
              toast.success(`State: ${state}, Native Language: ${language}`);
              setToastShown(true); // Mark that the toast has been shown
            }
          } catch (err) {
            setError("Error fetching state name: " + err.message);
```

```
        }
      };

      getLocation();
    }, [toastShown]); // Effect depends on toastShown to avoid multiple toasts

    return (
      <div className="geo">
        {stateName && <p>State: {stateName}</p>}
        {nativeLanguage && <p>Native Language: {nativeLanguage}</p>}
        {error && <p style={{ color: 'red' }}>{error}</p>}

        {/* ToastContainer is required to display the toast notifications */}
        <ToastContainer />
      </div>
    );
};

export default Location;
```

APPENDIX-B
SCREENSHOTS

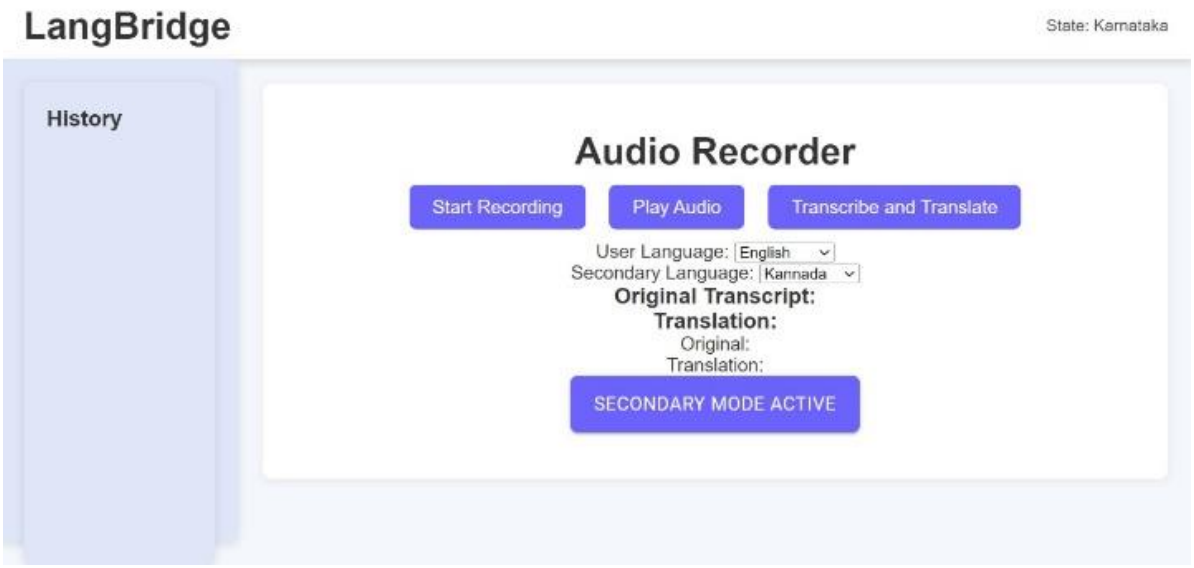


Fig B5: Final setup of the project

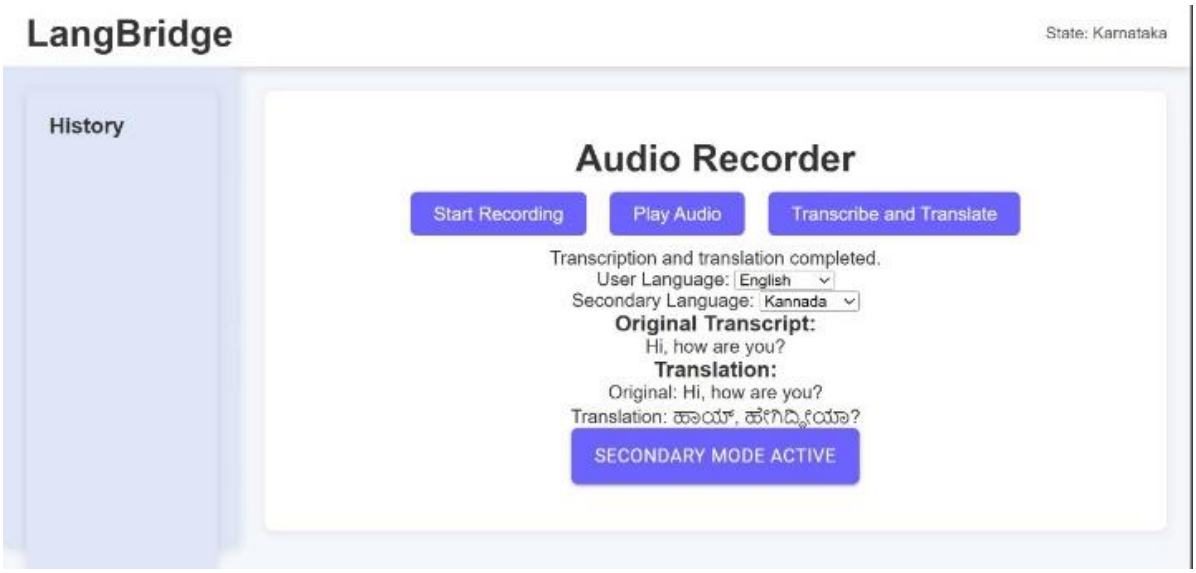


Fig B6: Final output of the project

APPENDIX-C

ENCLOSURES

Sustainable Development Goals:

Sustainable Development Goals of Real-Time Language Translator:

The Sustainable Development goals of creating a web application that focus on generating a Real-Time translation of the given audio into the necessary translated text can be as noted as follows.

Quality Education (SDG 4):

By providing real-time language translation, the project can make educational resources accessible to a wider audience, including those who speak different languages or have hearing impairments.

Industry, Innovation, and Infrastructure (SDG 9):

The integration of advanced AI technologies like Whisper AI, ASR, and NMT promotes innovation and the development of robust infrastructure for real-time communication and translation.

Reduced Inequalities (SDG 10):

The project helps bridge language barriers, promoting inclusivity and reducing inequalities in access to information and services.

Peace, Justice, and Strong Institutions (SDG 16):

By facilitating communication across different languages, the project can support peaceful and inclusive societies, enhance access to justice, and build effective, accountable institutions.

Partnerships for the Goals (SDG 17):

The project encourages collaboration between different stakeholders, including technology providers, educational institutions, and governments, to achieve sustainable development goals.

GITHUB LINK : MohammedNihal6/CSE-G81